J Braz Comput Soc (2012) 18:299–310
DOI 10.1007/s13173-012-0072-8

WTI

Partially labeled data stream classification with the semi-supervised *K*-associated graph

João Roberto Bertini Jr. · Alneu de Andrade Lopes · Liang Zhao

Received: 15 June 2011 / Accepted: 22 March 2012 / Published online: 17 April 2012

Abstract Regular data classification techniques are based mainly on two strong assumptions: (1) the existence of a reasonably large labeled set of data to be used in training; and (2) future input data instances conform to the distribution of the training set, i.e. data distribution is stationary along time. However, in the case of data stream classification, both of the aforementioned assumptions are difficult to satisfy. In this paper, we present a graph-based semi-supervised approach that extends the static classifier based on the *K-associated Optimal Graph* to perform online semi-supervised classification tasks. In order to learn from labeled and unlabeled patterns, here we adapt the optimal graph construction to simultaneously spread the labels in the training set. The sparse, disconnected nature of the proposed graph structure gives flexibility to cope with non-stationary classification. Experimental comparison between the proposed method and three state-of-the-art ensemble classification methods is provided and promising results have been obtained.

This is a revised and extended version of a previous paper that appeared at WTI 2010 (III International Workshop on Web and Text Intelligence) and has been recommended to JBCS.

J.R. Bertini Jr. (✉) · A.A. Lopes · L. Zhao
Instituto de Ciências Matemáticas e de Computação, USP, Avenida Trabalhador São-Carlense, 400, 13566-590 São Carlos, Brazil
e-mail: bertini@icmc.usp.br

A.A. Lopes
e-mail: alneu@icmc.usp.br

L. Zhao
e-mail: zhao@icmc.usp.br

Keywords Semi-supervised online classification · Incremental learning · Graph-based learning · Concept drift

1 Introduction

Recently, graph-based (also referred to network-based) algorithms applied to data mining tasks have attracted great attention in both theoretical research and practical applications [5]. This growing interest is mostly justified due to the advantages provided by graph representation, such as revealing topological structure of input data and the ability of identifying arbitrary shapes of data clusters [27]. In such graph-based algorithms, each vertex of the graph represents a data pattern (data instance) and the edges stand for some relation of similarity between vertices. In order to reveal significant relations within a data set, the following rule is usually considered for establishing connections between data patterns: the higher the similarity among data, the higher the probability of connection [39]. Stated in this way, nearby patterns tend to be heavily linked together while distant patterns may form a sparse structure. This property has been extensively explored using graph-based solutions, especially considering unsupervised tasks like clustering [32] and dimensionality reduction [1]. Only recently graph-based classification has been addressed, usually by the wrap of semi-supervised learning [38].

Semi-supervised learning methods concern the problem of automatic classification considering data sets with a small number of labeled data and a large amount of unlabeled data [7]. Such approach relies on the fact that labeled data are difficult to be gathered and often are associated with high costs, while unlabeled data are abundant in most applications and generally easy to be collected. Moreover, the manually labeling process is not always reliable or practicable.

For example, consider obtaining enough labeled data to train a classifier for a spam detection task (i.e. classifying spam and valid email). Such application design (1) incurs cost in paying an expert or a group of users to label what they call spam from what they consider real email; (2) may result in inconsistencies if we accept all human categorization. For instance, an email message may be considered as a spam by some people, but it may be considered as a valid email by others; (3) not to mention the time required to manually label enough data to train a regular supervised learning method.

A spam detection application is really a stream classification problem, in the sense that the classifier needs to classify new patterns at the time they arrive [35]. In this kind of applications, the underlying data distribution changes over time, and such changes often make the model built on old data inconsistent to the newly arrived data. This problem, known as concept drift [34], requires frequent updating of the model. Summarizing, we have a classification problem which consists of a data stream where few instances are labeled and data distribution may change over time. This scenario poses a challenging task for machine learning because it presents too few labeled data along the stream to apply a supervised incremental algorithm and the presence of concept drift disables the use of static classifiers. In fact, only recently such applications have been properly addressed due to the concept of learning through both labeled and unlabeled data and the development of semi-supervised learning strategies.

In the development of semi-supervised learning algorithms, many efforts have been made on the use of a clustering algorithm to group the patterns and further spread the labels. When considering this approach, the K-means algorithm is a natural choice. Li et al. [20] proposed a tree-based algorithm which uses the K-means to spread labels at the leaves of a tree. Masud et al. [22] proposed an ensemble of micro-clusters, obtained by using the K-means algorithm, then instances are classified according to the K-nearest neighbor rule. Ditzler and Polikar [11] proposed an ensemble of classifiers, named WEA, which are trained with labeled patterns only. Then, unlabeled data and the K-means algorithm are used to generate a mixture of Gaussian models for further adjusting the weights of each classifier. Zhang et al. [37] use the semi-supervised SVM [8] allied to a version of the K-means, referred to as relational K-means, to construct new features to the labeled examples by using information extracted from unlabeled instances. Some investigations have been made to tackle specific problems, e.g. Erman et al. [12] proposed a method to perform traffic classification in computer networks with partially labeled data. Their method uses a clustering algorithm, such as K-means, to obtain the clusters and then, the labels are spread using the maximum likelihood estimation. The clusters that remain unlabeled are likely to be an undefined group. Also regarding computer network, Yu et al. [36] have considered the problem of intrusion detection. They employ a strategy similar to the K-means by grouping the labeled data and then, the labels are spread to the whole data set according to the distances from the clusters to unlabeled patterns. Finally, a SVM is trained to detect intrusion.

To the best of our knowledge, graph-based approach has not been considered to tackle streaming classification problems where data are partially labeled; although it is successfully applied to semi-supervised learning, especially to the transduction problem [2, 6, 10, 25]. In view of the recent developed graph-based nonparametric classification method and its good performance on stationary data sets [4, 21]; we had proposed a non-stationary version with initial results reported in Ref. [3]. In this paper, we propose an extended version to be applied in the context of non-stationary stream of partially labeled data. The aforementioned graph-based method is based on representing the training set as a special graph, referred to as K-associated graph. The K-associated graph is able to represent similarity relations among data instances and the purity of a component (connected subgraph) is able to represent the data topology. Purity characterizes the degree in which instances of different classes are mixed in a same region of the data space. In this work we propose a new constructing procedure for the K-associated graph that takes into account partially labeled sets. Also, this work shows how the graph is updated along the time to allow data stream processing.

The remainder of the paper is organized as follows: In Sect. 2, we briefly describe the problem of concept drift and also a toy example to illustrate a scenario where incremental learning is applicable. Section 3 presents the proposed method for non-stationary partially labeled stream classification. This section is further divided into four subsections, where Sect. 3.1 first introduces the K-associated graphs and the K-associated optimal graph. The new method for constructing the aforementioned graphs from partially labeled data sets is described in Sect. 3.2. Moreover, Sect. 3.3 briefly treats the static KAOG classifier [4] and Sect. 3.4 details how the graph is updated over time. Section 4 presents the experimental results concerning the performance comparison between the proposed algorithm and three well-know fully supervised streaming ensemble classifiers on non-stationary partially labeled benchmarks. Section 5 concludes the paper and discusses some future works.

2 Background

The non-stationary nature of data streams cause a phenomenon called *concept drift* (or also termed *concept substitutions*, *revolutionary changes*, *population drift*) [17, 24, 28, 34], in which concept refers to the data distribution in a

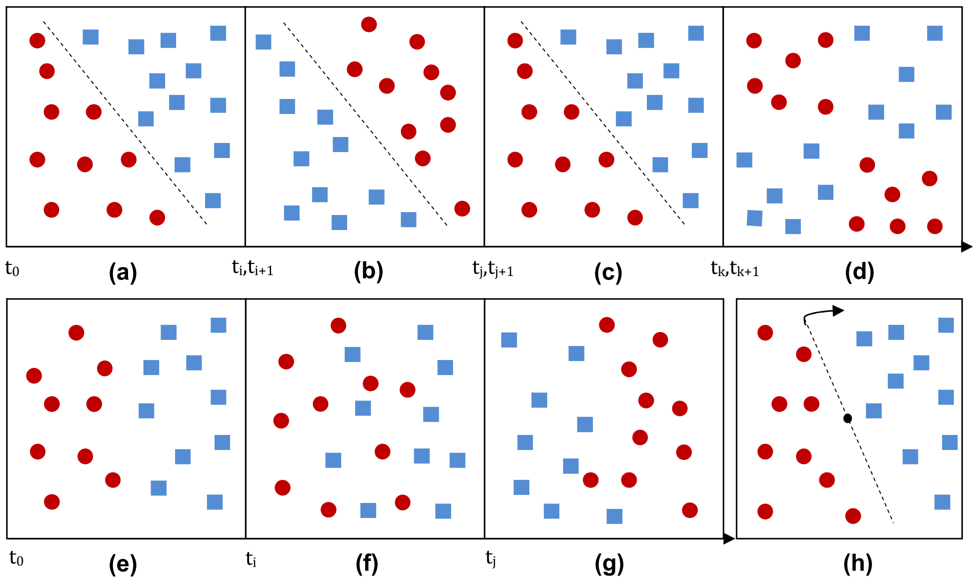

Fig. 1 Frequent concept drift characterizations, (**a**)–(**d**), abrupt drifts with recurrence; (**e**)–(**g**) gradual drift and (**h**) example subjugated to the velocity it is modified

given period of time. In the literature, however, the term concept drift has been used in reference to different phenomena relating to drop down the classifier accuracy performance [31]. According to Kelly et al. [17], concept drift occurs due to alterations on the following probabilities of data production:

– *A priori* of classes $P(\omega_1), \dots, P(\omega_M)$, i.e. alteration on the relative size of a given class or the appearance of new classes.
– Conditional $P(\mathbf{x} \mid \omega_i)$, i.e., changing on class definition. For example, changes in the shape of a class.
– Conditional *a posteriori* $P(\omega_i \mid \mathbf{x})$, i.e., modification on some of the attributes;

In general terms, concept drift can be characterized according to the variation of the concepts mainly regarding two features, *velocity* and *recurrence* along the time. Basically, in the former, a concept drift can be divided into *gradual drift* and *abrupt drift*; while in the latter, a concept drift is *recurrent* if past concept turns to be current concept. Both kinds of concept drift are sketched in Fig. 1.

In Fig. 1, the (blue) rectangles represent the instances that belong to class ω_1 and the (red) circles represent the instances belonging to class ω_2. Consider Figs. 1(a)–(d) as a sequence of data distributions of an application presented in time, initiating at t_0. The concept drifts that occur between distributions of Figs. 1(a) and 1(b), as well as between Figs. 1(b) and 1(c), are abrupt. Also notice that the distribution shown by Fig. 1(c) is similar to that in Fig. 1(a), which mean that the distribution at time t_0 in Fig. 1(a) occurs again at time t_{j+1}, after experiencing a completely different distribution (Fig. 1(b)). This phenomenon characterizes a recurrent concept. As the time line shows, from Figs. 1(a) and 1(d), each distribution can, eventually, remain static for a given period of time, e.g., the initial distribution remains static from t_0 to t_i. Nonetheless, on the next iteration t_{i+1}, the distribution can be totally altered, i.e. an abrupt drift occurs. The drift between distributions in Figs. 1(c) and 1(d) is also considered abrupt, in spite of being less severe than the previous one. Consider now a situation where two groups of data from different classes cross each other along time, depicted in order in Figs. 1(e)–(g), from an initial distribution (Fig. 1(e)) to a final one (Fig. 1(g)) with Fig. 1(f) corresponding to an intermediate distribution. In such a scenario, the distribution varies smoothly throughout the time, which characterizes a gradual drift. At last, let Fig. 1(h) represent a distribution determined by a rotating hyperplane along the time. If the hyperplane is rotated by $\pi/4$ regularly at a given period of time, the drift is characterized as gradual and recurrent at every eight alterations of the hyperplane. However, if the angular velocity rate is increased, say to π, the drift now can be considered abrupt. This demonstrates that it is surprisingly difficult to accurately characterize concept drifts considering only velocity and recurrence. In view of this problem, many researchers have proposed different drift categories; for a recent work, refer to Ref. [23].

In spite of characterizing concept drift, the main concern is that, most of the time, the variation in the underlying data distributions degenerates the performance of the classifier in use. The need for replacing a classifier due to the drop in accuracy, caused or not by a concept drift, is called *virtual concept drift* [18]. The trivial way to treat virtual drift is to replace the low accuracy classifier by a new one. However, such strategy brings at least three prohibitive drawbacks, (i) retraining new classifiers usually is computationally expensive; (ii) detecting when the current classifier is no longer useful is quite challenging, mainly due to the natural fluctuations in performance that can be confused with real concept

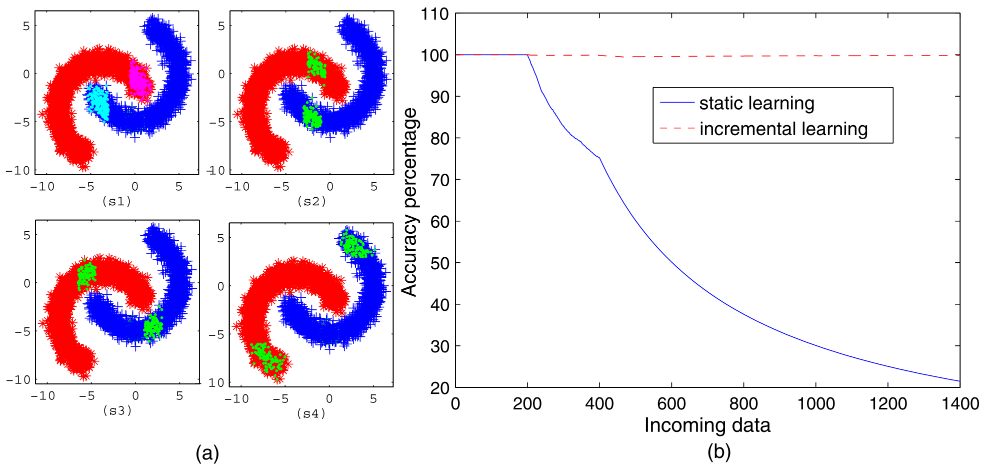

Fig. 2 (**a**) Artificial data set (Banana set) divided into sequential groups for simulating a non-stationary domain. The highlighted examples represent those chosen examples in a run. (**b**) Cumulated accuracy for static and incremental learning using the KAOG classifier

drift; (iii) selecting what data should be used to train the new classifier is also a hard task. Fortunately, incremental learning algorithms can be applied to provide practical solutions to tackle classification problems on non-stationary domains. Such an approach enables a classifier to acquire knowledge during application phase, updating the model with new data, and without explicitly retraining itself [14, 30].

For clarifying the advantages of an incremental classifier over a static one in non-stationary domains, consider the following experiment with the artificial data set known as banana set (see Fig. 2(a)). The experiment consists of comparing both approaches, static and incremental, of the classifier based on the K-associated graph. For doing so, in both cases, the classifier is trained with a limited subset (set "s1" in Fig. 2(a)). The rest of the set is divided into seven groups and used as test sets that are sequentially presented to the classifier. In Fig. 2(a) "s1" is the original training set and the others correspond to the first, second, and seventh test sets, respectively. The results are averaged over 10 runs, at each run, an optimal graph is built considering 400 examples (200 of each class) randomly chosen from the training data group. After training, the test examples are chosen obeying the group sequence, one-by-one, 200 examples are randomly chosen from each group (100 for each class), then, the next group takes place and so on.

Figure 2(b) shows the results of the comparison between the K-associated static and incremental classifiers. The significant difference between them is due to the fact that the static classifier no longer learns with new instances, however the incremental classifier is able to learn during classification phase. The presented incremental learning process is analogous to the linearization technique widely used to study local properties of non-linear systems. Specifically, linearization of a neighborhood of a certain point corresponds to subset selection in incremental learning. Nonlinearity of the system corresponds to twisted shape of classes and changing of data distribution over time. In a non-linear systems, linearization usually can obtain good approximation if the neighborhood under analysis is small. For the same reason, we expect that good classification results can be obtained by updating the network with small data subset each time.

3 The semi-supervised K-associated optimal graph

The semi-supervised K-associated graph, proposed here, consists of a modification of the K-associated graph [4] to deal with both labeled and unlabeled data during the graph construction procedure. Therefore, in order to introduce the semi-supervised version, a brief revision of the K-associated graph is presented in Sect. 3.1. It is followed by the semi-supervised K-associated graph construction presented in detail in Sect. 3.2. Both supervised and semi-supervised K-associated optimal graphs can be seen as the training process for the KAOG classifier which uses the components of the graph and their purities to classify new data instances, as will be exposed in Sect. 3.3.

3.1 The K-associated graph and the K-associated optimal graph

A K-associated graph is constructed from a vector-based data set $X = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ by representing each data instance $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{ip}, c_i)$ as a vertex v_i with its associated class label c_i, where $c_i \in \Omega = \{\omega_1, \omega_2, \dots, \omega_M\}$ and M is the number of classes in the problem. The graph construction resembles to a KNN graph, due to the use of a predefined number of neighbors, K, that each vertex must connect. Although the K-associated graph does differ from the KNN approach by the fact that amongst the possible K neighbors of a vertex v_i, it can only be connected to neighbors of the same class as v_i. Hence, we consider the label-independent and the label-dependent K-neighborhood of vertex v_i. The former is simply the set of vertices that represents the K nearest neighbors of the instance $\mathbf{x}_i$ according to a given measure and will be noted by $\Lambda_{v_i,K}$. The latter comprises only the vertices with the same class as v_i among its K nearest neighbors, and is defined as $\Delta_{v_i,K} = \{v_j \mid v_j \in \Lambda_{v_i,K} \text{ AND } c_i = c_j\}$.

In a formal way, the K-associated graph is defined as a directed graph $G = (V, E)$ which consists of a set of labeled vertices V and a set of edges E between them, where an edge $e_{ij} = (v_i, v_j)$ connects vertex v_i with vertex v_j if and only if $v_j \in \Delta_{v_i,K}$. As a consequence, only vertices of the same class can be connected. The resulting K-associated graph can be viewed as a set of disjoint subgraphs or *components* $C = \{C_1, \ldots, C_\alpha, \ldots, C_R\}$. Each component C_α is composed by vertices of a single class, thus each component represents a single class, which we refer to the label of component C_α as $\hat{C}_\alpha$. The number of components R varies according to the magnitude of K, but always lies in the range $N \geq R \geq M$, with N being the number of vertices in the training set and M the number of classes. Higher values of K induce fewer and larger components in the constructed graph, while lower values lead to small sized ones. This wire mechanism leads to a graph with some important features: (i) By varying K, different graphs can be generated, and as the value of K increases, the number of components decreases monotonically to the number of classes. (ii) The total number of edges among the vertices of a component C_α is proportional to K and can be at most equal to KN_α, where N_α is the number of vertices in component C_α. (iii) This maximum value is only achieved if all vertices in the neighborhood of any vertex of the component have the same class. Likewise, nearby vertices of other classes decrease the number of connections of the given component. Thus, one can define a measure of "purity" for components, as explained ahead.

Let the degree d_i of a vertex v_i be defined as the sum of the connections it receives (in-degree) and the connections it performs (out-degree) to other vertices, so $d_i = d_i^{\text{in}} + d_i^{\text{out}}$. Also, consider the average degree taken for component C_α be defined by $D_\alpha = 1/N_\alpha \sum_{v_i \in C_\alpha} d_i$. According to the way that the K-associated graph is constructed, a vertex can perform at most K connections, thus, the maximal total out-degree of component C_α is KN_α; symmetrically, the total in-degree is also KN_α, resulting in average degree being equal to $2K$. Hence, a key idea is to use the ratio defined in Eq. (1) as a measure of "purity" for component C_α, because it quantifies how intertwined a component is with vertices of other classes,

$$\Phi_\alpha = \frac{D_\alpha}{2K} \tag{1}$$

In this way, $\Phi_\alpha = 1$, if and only if, for every v_i in the component C_α, all the K neighbors have the same class label of v_i. On the other hand, if there exists noise or two or more classes are mixed together, vertices in this region are unable to make their K connections due to the existence of vertices of other classes in the neighborhood of some vertices. In the latter case, the more mixing the components are, the lower their average degrees D_α and consequently their respective purities Φ_α are.

Clearly, the structure of a K-associated graph depends on the value of K and on the nature of the input data set. Also, K-associated graphs formed with different K will present different components with different purity values. Bearing this in mind, a suggestive idea is to obtain a graph with the best organization of components without using a unique value of K, i.e., each component has its own optimal value of K, denoted as K_α for component C_α. Therefore, the rationale for obtaining the optimal graph is to construct $K = 1, \ldots, K_{\max}$ associated graphs while keeping the best components found at each K throughout this process. Let β also be an index of component, therefore, a component $C_\beta^{(K+z)}$ from the $(K + z)$-associated graph will replace all components from the K-associated graph that satisfy Eq. (2), for some integer $z \geq 1$ and $(z + K) \leq K_{\max}$,

$$\Phi_\beta^{(K+z)} \geq \Phi_\alpha^{(K)} \quad \text{for all } C_\alpha^{(K)} \subseteq C_\beta^{(K+z)} \tag{2}$$

The optimal graph improves the representation of the training set and provides the best configuration of components according to their purities. It corresponds to the best graph organization regarding the purity measure.

3.2 The semi-supervised K-associated optimal graph

Consider now obtaining the optimal graph from a partially labeled set X. It is easy to see that it is not possible to obtain the aforementioned graph through the previous description due to the presence of unlabeled patterns. Therefore, we propose here the semi-supervised construction of the K-associated optimal graph.

The problem addressed here regards the absence of enough labeled data in a given data set to employ a regular supervised method. Therefore, it is necessary to consider a semi-supervised method in order to induce a classifier from both labeled and unlabeled patterns. Hence, consider the data set $X = \{(\mathbf{x}_1, c_1), \ldots, (\mathbf{x}_l, c_l), \mathbf{x}_{l+1}, \ldots, \mathbf{x}_N\}$ with l labeled patterns $(\mathbf{x}_i, c_i)$ and $N - l$ unlabeled patterns $\mathbf{x}$ (or $(\mathbf{x}_j, \emptyset)$). As its supervised counterpart, the semi-supervised K-associated optimal graph construction involves creating a sequence of semi-supervised K-associated graphs. The main difference between the supervised and semi-supervised K-associated graphs can be stated in relation to the set of neighbors, to which each vertex connects. Instead of considering only the label-dependent set $(\Delta_{v_y,K})$, here, each vertex v_i connects to all vertices in the set $\Gamma_{v_i,K} = \{v_j \mid v_j \in \Lambda_{v_i,K} \text{ AND } (c_j = c_i \text{ OR } c_i = \emptyset \text{ OR } c_j = \emptyset)\}$. This set encompasses the K nearest neighbors of v_i whose classes are not different from the class of v_i. This means that, among its K nearest neighbors, v_i connects to those vertices which belong to the same class of v_i or to those with no label. If v_i itself does not have a class label, it connects to all the K nearest neighbors without considering their classes.

As a consequence of connecting unlabeled vertices to labeled vertices regardless to their classes, components with more than one class may be formed. However, having components with more than one class precludes the classifier to make decisions. In other words, each component must be formed by vertices belonging to a single class and different from null. Thus, to overcome this problem, we propose splitting those components with vertices associated to two or more classes. For splitting a component, the rationale is to cut a few edges in order to end up with separated well-connected clusters of vertices. In other words, this is a min-cut problem, which can be resolved, for example, by the Ford–Fulkerson algorithm [9] for the two-class case. However, as we consider multi-class classification, there exists the problem that a component might be composed by vertices from more than two classes. For this reason, we propose cutting the component based on the purity of vertex, defined as $d_i/2K$, where d_i stands for the degree of v_i. Again, consider $W_{i,j}$ the distance, used to construct the graph, between patterns $\mathbf{x}_i$ and $\mathbf{x}_j$. Thus the proposed separation approach consists of successive removing the edge with minimum value of cut from the component, as defined in Eq. (3). Otherwise stated, the next edge to be removed $(v_a, v_b) \in C_\alpha$ must satisfy $\text{cut}_{a,b} = \min(\text{cut}_{i,j})\ \forall (v_i, v_j) \in C_\alpha$.

$$\text{cut}_{i,j} = \min\left(\frac{d_i}{2K}, \frac{d_j}{2K}\right)\frac{1}{W_{i,j}} \tag{3}$$

The cutting process in the component C_α finishes until it is separated into single class components. The rationale behind the criterion is that by cutting the edges that connects low purity vertices and whose respective patterns are distant from each other, it is more likely to obtain separated well-connected components. In fact, low purity vertices are usually found in boundary regions between components of different classes in supervised tasks. However, in the semi-supervised scenario, purity itself can be a misleading measure due to high connection probability of the unlabeled vertices. Therefore the distance weight in Eq. (3) favors cutting the edges with highest distance in the component.

Algorithm 1 details the construction of the semi-supervised K-associated optimal graph. The function *findComponents*() determines the graph components by implementing a breadth-first search [9]. Then, the components having vertices belonging to more than one class are separated by the function *splitNonSingleClassComponents*(). This function implements the cutting procedure described earlier and returns two or more single class components, which can include components without a class label. The next step consists of spreading the labels within every component by calling the function *spreadLabel*(). After this

Algorithm 1 Semi-supervised K-associated Optimal Graph construction from partially labeled set—KAOGSS

```
Input: $X = \{(\mathbf{x}_1, c_1), \ldots, (\mathbf{x}_l, c_l), \mathbf{x}_{l+1}, \ldots, \mathbf{x}_N\}$
Symbols: $G_s^{(K)}$—$K$-associated Graph built from labeled and unlabeled patterns
    $G_s^{(\mathrm{opt})}$—$K$-associated Optimal Graph built from labeled and unlabeled patterns
    $\Gamma_{v_i,K}$—set of $K$ nearest neighbors of $v_i$ within the same label or no label
    $R$—number of components in the current $K$-associated graph $G_s^{(K)}$
    $M$—the number of classes in $X$
1: $K \Leftarrow 1$
2: repeat
3:     $C \Leftarrow \emptyset$
4:     $G_s^{(K)} \Leftarrow \emptyset$
5:     for all $v_i \in V$ do
6:         $\Gamma_{v_i,K} \Leftarrow \{v_j \mid v_j \in \Lambda_{v_i,K}$ and $(c_j = c_i$ or $c_i = \emptyset$ or $c_j = \emptyset)\}$
7:         $E \Leftarrow E \cup \{e_{ij} \mid v_j \in \Gamma_{v_i,K}\}$
8:     end for
9:     $C \Leftarrow findComponents(V, E)$
10:    $C \Leftarrow splitNonSingleClassComponents(C)$
11:    for all $C_\alpha \in C$ do
12:        $C_\alpha \Leftarrow spreadLabel(C_\alpha)$
13:        $\Phi_\alpha \Leftarrow purity(C_\alpha)$
14:        $G_s^{(K)} \Leftarrow G_s^{(K)} \cup \{(C_\alpha(V', E'); \Phi_\alpha)\}$
15:    end for
16:    if $K = 1$ then
17:        $G_s^{(\mathrm{opt})} \Leftarrow G_s^{(K)}$
18:    else
19:        for all $C_\beta^{(K)} \subset G_s^{(K)}$ do
20:            for all $C_\alpha^{(\mathrm{opt})} \subseteq C_\beta^{(K)}$ do
21:                if $(\Phi_\beta^{(K)} \geq \Phi_\alpha^{(\mathrm{opt})}$ or $\hat{C}_\alpha^{(\mathrm{opt})} = \emptyset)$ then
22:                    $G_s^{(\mathrm{opt})} \Leftarrow G_s^{(\mathrm{opt})} - \bigcup_{C_\alpha^{(\mathrm{opt})} \subseteq C_\beta^{(K)}} C_\alpha^{(\mathrm{opt})}$
23:                    $G_s^{(\mathrm{opt})} \Leftarrow G_s^{(\mathrm{opt})} \cup \{C_\beta^{(K)}\}$
24:                end if
25:            end for
26:        end for
27:    end if
28:    $K \Leftarrow K + 1$
29: until $R = M$
30: Output: The $K$-associated optimal graph within all vertices with labels $G^{(\mathrm{opt})} = \{C_1^{(\mathrm{opt})}, \ldots, C_\alpha^{(\mathrm{opt})}, \ldots, C_R^{(\mathrm{opt})}\}$ where component $C_\alpha^{(\mathrm{opt})} = (G'(V', E'); \Phi_\alpha, K_\alpha)$
```

stage, all vertices in any given component are labeled with a single class label or are unlabeled. To finish the K-associated graph construction, the purity measure is calculated through the function *purity*() for all components. At the end of this process, if $K = 1$, then the graph generated so far is the optimal graph and it is assigned to $G_s^{(\mathrm{opt})}$. Otherwise, each component of the current K-associated graph, $G_s^{(K)}$, is compared to the components in the graph, $G_s^{(\mathrm{opt})}$, having the same vertices (condition in line 20). The new component will substitute the corresponding old ones if the purity is increased or maintained for the labeled components. The process goes on by increasing K and generating a new K-associated graph until the number of components in this new graph matches the number of classes in the problem ($R = M$).

In summary, the main modifications in the original K-associated optimal graph construction algorithm [4] include connecting each vertex to all its neighbors with the same class or without a class label (line 6) and merging every component with empty class (in the Algorithm 1, $\hat{C}_\alpha$ stands for the class of a component) to another component, independent of purity. Notice that the present algorithm not only can construct the K-associated optimal graph, but also, by doing so, can spread the labels throughout the whole training set. Therefore, the KAOGSS algorithm is a transductive method.

3.3 The KAOG classifier

This section presents the nonparametric classifier that uses the K-associated optimal graph structure to infer the class of new patterns, for more details, please refer to Ref. [4]. In order to present how a new pattern is classified, consider again a training pattern $\mathbf{x}_i$ represented by $\mathbf{x}_i = (x_{i1}, x_{i2}, \dots, x_{ip}, c_i)$, which $\mathbf{x}_i$ represents the ith training pattern with c_i its associated class label, in a M-class problem $c_i \in \Omega = \{\omega_1, \omega_2, \dots, \omega_M\}$. In the same way, a new pattern is defined as $\mathbf{y} = (y_1, y_2, \dots, y_p)$, excepted that now its class label must be estimated. Consider also the set of components of the optimal graph $C = \{C_1, \dots, C_\alpha, \dots, C_R\}$, where R is the number of components and $R \ge M$. In order to classify the new pattern $\mathbf{y}$, we must firstly transform it to a vertex, noted by v_y, then connect it into the graph as explained ahead. Consider K_L the largest value of K in the K-associated optimal graph, or equivalently the K value from the last obtained component. For every new pattern $\mathbf{y}$, we do:

```
1. Calculate the distances between the new pattern $\mathbf{y}$ and all
   elements $\mathbf{x}_i$ in the training set
2. Find the $K_L$ nearest neighbors of $\mathbf{y}$; noted in ascending
   order as $\bar{\Lambda}_{v_y,K_L} = \{\mathbf{x}_{(1)}, \mathbf{x}_{(2)}, \dots, \mathbf{x}_{(k)}, \dots, \mathbf{x}_{(K_L)}\}$
3. For $k = 1$ to $K_L$
      Locate the vertex (and component) that represents
      $\mathbf{x}_{(k)}$, say $v_j \in C_\alpha$
      If $k \le K_\alpha$ then
          Connect $v_y$ to $v_j$
```

Once the new vertex v_y is connected to the K-associated graph, its class label is estimated using the Bayes theory [15]. The connection established during classification are temporary, i.e. they will not be incorporated into the graph structure. The *posterior* probability of a new vertex v_y to belong to component C_α given the set of label-independent neighbors of v_y, noted by Λ_{v_y}, is defined by Eq. (4),

$$P(v_y \in C_\alpha \mid \Lambda_{v_y}) = \frac{P(\Lambda_{v_y} \mid v_y \in C_\alpha) P(v_y \in C_\alpha)}{P(\Lambda_{v_y})} \tag{4}$$

Knowing that each component C_α has been formed in a particular K-associated graph among the various generated graphs, we must consider the particular value of K in which C_α was formed, noted by K_α. Let Λ_{v_y,K_α} represent the set of K_α nearest neighbors of v_y. Thus, in order to estimate the probability $P(\Lambda_{v_y} \mid v_y \in C_\alpha)$, one must consider the fraction among the connections made with component C_α over all possible K_α connections, as shown in Eq. (5),

$$P(\Lambda_{v_y} \mid v_y \in C_\alpha) = \frac{|\{\Lambda_{v_y,K_\alpha}\}|}{K_\alpha} \tag{5}$$

The *prior* probabilities $P(v_y \in C_\alpha)$ are defined as the normalized purities among the components to which v_y is connected as $P(v_y \in C_\alpha) = \Phi_\alpha / \sum_{N_{v_y,C_\beta} \neq 0} \Phi_\beta$, where N_{v_y,C_β} represents the number of connections v_y has to component C_β. Accordingly, the normalizing term is given by Eq. (6),

$$P(\Lambda_{v_y}) = \sum_{N_{v_y,C_\beta} \neq 0} P(\Lambda_{v_y} \mid v_y \in C_\beta) P(v_y \in C_\beta) \tag{6}$$

In many cases, there are more components than number of classes, according to Bayes optimal classifier, it is necessary to sum the *posterior* probabilities of all components corresponding to the same class. Finally the largest value among the found *posterior* probabilities reflects the most probable class for the new pattern, according to Eq. (7), where $\varphi(\mathbf{y})$ stands for the class attributed for instance $\mathbf{y}$,

$$\varphi(\mathbf{y}) = \arg\max\{P(\mathbf{y} \mid \omega_1), \dots, P(\mathbf{y} \mid \omega_M)\} \tag{7}$$

3.4 Classifying partially labeled data stream

This section exposes how the proposed graph-based structure copes with non-stationary classification. Consider a stream $S = \{X_1, Y_1, \dots, X_T, Y_T\}$, where $X_t = \{(\mathbf{x}_1, c_1), \dots, (\mathbf{x}_l, c_l), \mathbf{x}_{l+1}, \dots, \mathbf{x}_N\}$ contains labeled and unlabeled patterns; while $Y_t = \{\mathbf{y}_1, \dots, \mathbf{y}_M\}$ is formed with unlabeled patterns only. Such streams may present concept drift at any time. Therefore, an online classifier should have the ability to evolve by adding new knowledge along time without being retrained. In the proposed approach, this dynamical evolution is done by considering a dynamic graph, named *principal graph*, which grows with the frequent addition of components provided by the K-associated optimal graph formation (Algorithm 1) along the data stream processing. Algorithm 2 details the proposed approach.

Algorithm 2 presents the KAOGINCSSL algorithm, which processes a data stream S composed of partially labeled and unlabeled data sets. The function *nextChunk*(S)

Algorithm 2 Incremental Algorithm KAOGINCSSL

```
Input: $S = \{X_1, Y_1, \ldots, X_T, Y_T\}$ {Data stream}
    $X_t = \{(\mathbf{x}_1, c_1), \ldots, (\mathbf{x}_l, c_l), \mathbf{x}_{l+1}, \ldots, \mathbf{x}_N\}$ {Partially labeled set}
    $Y_t = \{\mathbf{y}_1, \ldots, \mathbf{y}_M\}$ {Unlabeled set}
    $\tau$ {Forgetting parameter—set by user}
Symbols: $Z$—variable used to represent the next set to be processed, which may be partially labeled or unlabeled
    $G_P$ {Principal graph};
    $KAOGSS()$ {Semi-supervised $K$-associated optimal graph builder};
    $Classifier()$ {Sect. 3.3}
1: $G_P \Leftarrow \emptyset$
2: repeat
3:     $Z \Leftarrow nextChunk(S)$
4:     if $isPartiallyLabeled(Z)$ then
5:         $G_P \Leftarrow G_P \cup KAOGSS(Z)$
6:     else
7:         for all $\mathbf{y}_j \in Z$ do
8:             $\varphi(\mathbf{y}_j) \Leftarrow Classifier(G_P, \mathbf{y}_j)$
9:         end for
10:        for all $C_\alpha \subset G_P$ do
11:            if $t_\alpha > \tau$ then
12:                $G_P \Leftarrow G_P - C_\alpha$
13:            end if
14:        end for
15:    end if
16: until $S = \emptyset$
17: Output: $\varphi(\mathbf{y}_j)$—Estimated label for all unlabeled test pattern $y_j \in Y_t, t = 1, \ldots, T$.
```

removes the next set from stream S and put it into the variable Z used to represent a chunk of data. After assigning the next set to the variable Z, the algorithm determines if the set is partially labeled to be considered for training/updating (i.e. if the set has enough labeled patterns, e.g., at least 5 %) through the function *isPartiallyLabeled*(Z) which returns "true" if Z is partially labeled and "false" otherwise.

Therefore, the tasks of the algorithm are twofold, (i) incorporate new knowledge from both labeled and unlabeled patterns to subdue concept drift and (ii) predict the label for the unlabeled patterns presented in unlabeled sets. In the former task, the objective is to incorporate new knowledge from the recent obtained partially labeled set, thus a semi-supervised K-associated optimal graph is derived using Algorithm 1 (KAOGSS). As explained in Sect. 3.2, the KAOGSS algorithm generates the K-associated optimal graph spreading the labels to all vertices and the resulting graph is composed of several disjoint components. These new components are then merged to the principal graph (G_P), which is composed of independent components. However, the addition of new components increases the size of the principal graph, which may increment classification error and time. To avoid this problem, the principal graph should not grow unlimitedly, thus, old and unused components should be removed.

The task of classifying new patterns takes place if the set at hand is unlabeled, and it is resolved by simply applying the KAOG classifier using the principal graph to classify unlabeled vertices, as presented in Sect. 3.3. Component removal takes place during classification phase by applying a method named disuse rule. This rule establishes a maximum number of consecutive classifications in which a component is allowed to be unused (i.e. do not receive any connections during classification). The maximum value accepted is set by the parameter τ. When a component remains out of use after τ patterns are classified, it is removed from the principal graph. The algorithm finishes when the whole stream has been processed, i.e. $S = \emptyset$.

An important feature of stream classification algorithms is its ability to process data in a reasonable time, which includes the tasks of training, updating and classifying. The proposed algorithm consists of the following phases of data processing: (i) training or updating the principal graph, (ii) classifying new data and, (iii) removing unused components.

In the first phase, training or updating the principal graph is required whenever a partially labeled set X is presented. Let there be N instances in the set X; training (or updating) corresponds to build a semi-supervised K-associated optimal graph (Algorithm 1). As estimated in Ref. [4], the complexity order to build a supervised K-associated optimal graph is about $O(N^2)$—due to distance matrix calculation. Also, it has been shown that the K-associated optimal graph construction scales better than the C4.5 and the Gibbs Sampling algorithms. Taking into account that the only addition in processing time in the semi-supervised version is the need to verify whether a component presents more than one class and, in this case, the algorithm cuts out some edges to divide it into some single class components. Knowing that the process of finding and cutting a component by using the proposed technique depends on the number of edges and vertices in the component ($O(N_\alpha + E_\alpha)$), where N_α and E_α are the number of vertices and edges in the component C_α, respectively. Since K-associated graphs are sparse, thus, N_α or E_α is much smaller than the number of vertices in the whole graph. Allied to the fact that few components need to be partitioned (those components, which are composed of vertices from more than one class), it can be verified that the computational order of this phase remains $O(N^2)$.

Now we consider the second phase, the order of classifying a new pattern has also been estimated in the aforementioned work as $O(N_p)$, due to the distance calculation among the new vertex and the N_p vertices in the principal graph. Here, it is important to mention that there exist strategies for lowering the order, for example locating the nearest components firstly, instead of actually searching for the vertices neighbors. Such strategy decreases the computational cost to $O(N_{cp})$, with N_{cp} being the number of components in the principal graph, and $N_{cp} \ll N_p$. At last, component removal can be done by the disuse rule, which is done by simply checking the time parameter of each component, therefore, it has the order of $O(N_{cp})$.

4 Experimental results

The experimental results are obtained considering five non-stationary data sets, with three of them generated artificially, SEA [29], Sine and Circles [13] and the other two are real data, Spam and Elec2 [16]. For all the experiments, Algorithm 1 is used to spread the label to all the training sets.

In order to simulate a stream of partially labeled data and qualify how the algorithms react with different amount of labeled patterns, we have generated nine experiments for each domain, differing from each other regarding to the percentage of labeled patterns in the training sets. With the percentages of labeled patterns lying in the set {90 %, 80 %, 70 %, 60 %, 50 %, 40 %, 30 %, 20 %, 10 %}. Each stream is presented as a sequence of chunks of data, alternating between a partially labeled set and a fully unlabeled set. The partially labeled sets are used for training (or updating) the classifiers, while the fully unlabeled sets are used as test sets to estimate the classification accuracy of the algorithms. Here, we use the real labels of the test sets to estimate the classifier accuracy. Among the artificially generated stream data, the SEA domain is presented along 500 realizations of training sets with 60 patterns and tests set with 40 patterns. The other two streams, Circle and Sine, are presented along 200 realizations of alternating training and test sets, each of them with 25 patterns. Regarding the two real data sets, we consider a real situation where there are not enough data to use for testing. Therefore the same set is firstly used for testing and then for training. The Elec2 domain represents the electricity price fluctuation gathered during a given period (for details, please refer to Ref. [13]). The domain is composed of 45,312 patterns, which can be divided into 134 chunks of 336 patterns (except for the first set with 288), representing a week of price variation. The spam base is composed of 4601 patterns representing spam and real mail, the chunks, in this case, are defined with 45 patterns except for the initial with 101, and the stream is presented along 100 realizations.

Regarding the algorithms under comparison, three of them are ensemble algorithms chosen due to their high adaptability. The SEA ensemble [29] consists of a pool of C4.5 classifiers [26] and works by evaluating each of the decision trees, whose output is used to decide the ensemble output by a simple majority voting scheme. Every time a training batch arrives, a new decision tree is trained and it replaces the tree in the ensemble with the major number of mistakes up to that point. Another algorithm implemented for comparison is the DWM [19], which consists of an ensemble method that virtually can be composed by any classifiers. Briefly, the DWM algorithm adds a new incremental classifier to the ensemble every time an error is committed by the ensemble. Each single classifier has a weight that is decreased by a determined factor β every time it commits an error. For controlling the size of the ensemble, at every p iteration, those classifiers whose weight is less than a predefined threshold θ are removed. As recommended by the authors, the incremental Naive Bayes (see Ref. [19] for details and references) has been used as base classifier, therefore we note DWM-NB hereafter. The third algorithm, proposed by Wang et al. [33], is also an ensemble that uses a decision tree as base algorithm, similar to SEA, but with weighted classifiers. The weight of each base classifier is estimated by its classification accuracy in a test set. Therefore, the weight of base classifier h_k is given by $w_k = \mathrm{MSE}_r - \mathrm{MSE}_i$, where MSE_i corresponds to the generalization error and can be obtained through a cross-validation process; while MSE_r is the estimated error given the new data set, and can be calculated as $\mathrm{MSE}_r = \sum_{\omega_j \in \Omega} p(\omega_j)(1 - p(\omega_j))$, with $p(\omega_j)$ the percentage of instances belonging to class ω_j.

Figure 3(a) shows the accuracy for the tested algorithms on the nine different experiments regarding the percentage of labeled pattern in the training sets for the SEA domain. Each experiment result shows the classification accuracy on a test set averaged by 20 runs. Figures 3(b)–(d) show the results for the experiment with data sets with 20 % of labeled patterns. The results consist of the classification error rates for every presented test set, also averaged by 20 runs. The results of each algorithm under comparison (red curves) and the results of the proposed algorithm (blue curves) are put together and shown in Figs. 3(b)–(d).

Considering the experimental results displayed in Fig. 3(a), as expected, all the algorithms tend to degenerate their performance as the labeling percentage provided in the training sets decays. Notice that the proposed algorithm KAOGINCSSL and the DWM-NB algorithm have performed similarly throughout all the different label percentages domains, with exception to the experiment with 10 % of labeled patterns where the KAOGINCSSL algorithm presented a better performance. In fact, even when only 20 % of the training patterns are labeled, KAOGINCSSL and DWM-NB present similar performance, differentiating by the fact that the proposed algorithm is much more stable, presenting the smallest variance. Regarding the WCEA algorithm, from Fig. 3(a), we see that it is the algorithm that suffers the most as the amount of labeled patterns decreases. Again, when considering 20 % labeled set, in spite of presenting very close result for the average error percentage to the SEA algorithm performance, the WCEA algorithm presents a larger variation on error rate along the stream processing, as can be seem in Figs. 3(b)–(c). The SEA ensemble has the worst performance in this domain.

Now, consider the experiments on the other two artificial domains (Sine and Circle) and the two real domains (Elec2 and Spam), again, each with nine different realizations and classification results taken as the mean over all presented test sets averaged by 20 runs. Figure 4 presents the results.

As can be seen in Figs. 4(a)–(b), the proposed algorithm presents a large advantage on accuracy performance over

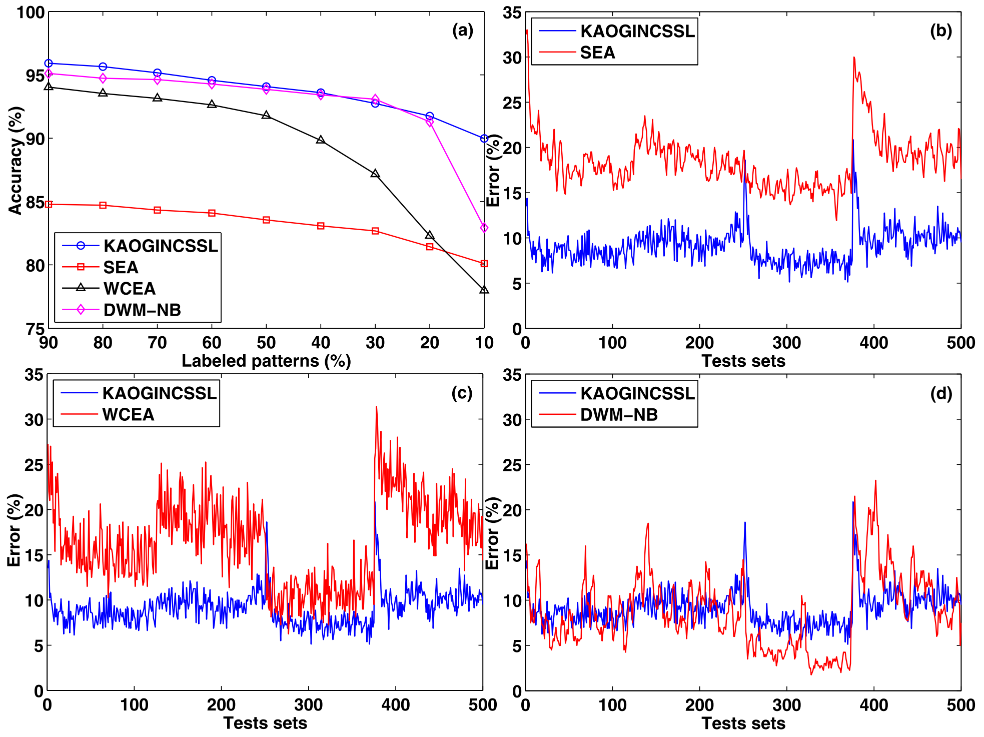

Fig. 3 Experiments with the SEA domain, (**a**) average accuracy for nine experiments with different percentages of labeled patterns, from 10 % to 90 %; and performance comparison through the SEA presentation with 20 % labeled patterns, between the KAOGINCSSL and (**b**) SEA, (**c**) WCEA and (**d**) DWM-NB

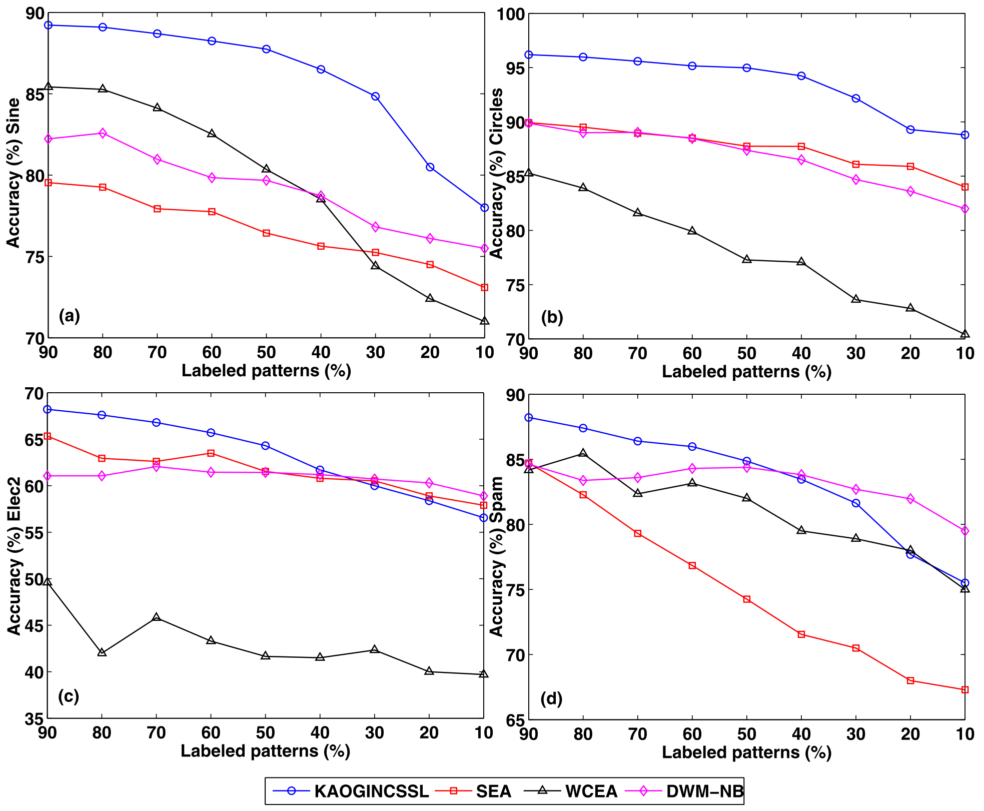

Fig. 4 Experiments with nine different percentages of labeled patterns in each data set for the domains Sine, Circle, Elec2 and Spam

the other algorithms, considering all the experiment configurations. This advantage, though, seems to decrease when real data sets are considered, which may be due to the fact that artificial domains are constructed in a controlled manner and therefore present some desired characteristics. On the other hand, real data sets present an unorganized scenario, e.g. concept drifts are not well-defined as in the artificial domains. Bearing this in mind, the advantage presented by

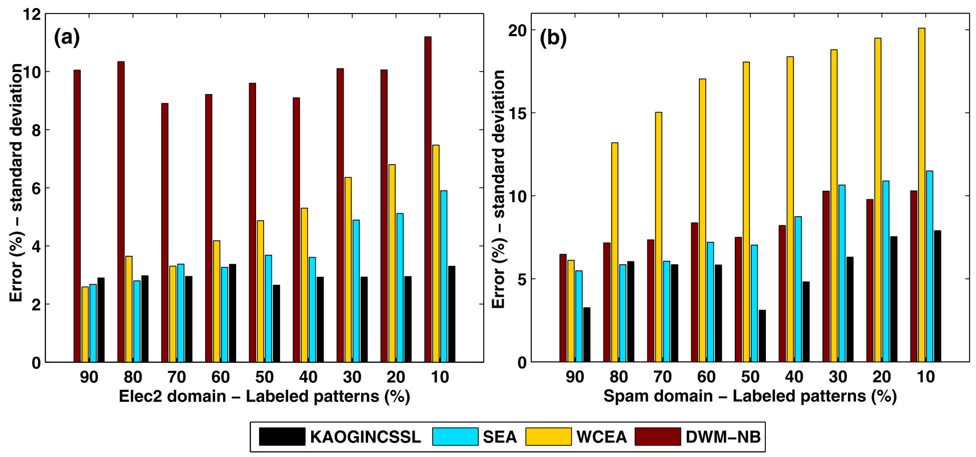

Fig. 5 Standard deviations for the four compared algorithms when processing the real domains (Elec2 and Spam) within the nine different percentage of labeled patterns in each data set

the proposed algorithm in the artificial domains can be partially explained due to the KAOGINCSSL ability to get rid of past concepts much faster than the ensemble algorithms used for comparison. Now, for better analyzing the real domains, consider also the standard deviation among the presented test sets and taken from a single run to best represent a real situation. Figure 5 presents the standard deviation for the nine considered experiments and the four compared algorithms when processing the real domains Elec2 and Spam.

In real applications, low variance or standard deviation is a desirable feature for a classifier, precisely the lower the standard deviation the more reliable is the classifier performance. Therefore, considering the results for the electricity domain presented in Fig. 4(c); except for the WCEA algorithm, all the others have presented a similar performance, in special for low levels of labeling (<40 %). Here, again the proposed algorithm obtained the best performances for the experiments with more than 50 % of labeled data. Analyzing the standard deviation in Fig. 5(a), it is easy to verify that the proposed algorithm presents the most reliable performance. The DWM-NB algorithm presents too higher values of standard deviation indicating high fluctuation in classification performance, in spite of presenting good average accuracy. The SEA ensemble has good accuracy results and low variance.

Regarding the results of the KAOGINCSSL algorithm in the Spam base shown in Fig. 4(d), at a first glance, almost the same trend as in the Elec2 domain can be observed. Because it has presented best average accuracy performance for experiments with more than 50 % labeled patterns and average performance for the rest. In spite of that, the KAOGINCSSL algorithm shows again the most regular performance as depicted in Fig. 5(b). The DWM-NB algorithm has also performs well, particularly up to the point where labeled data instances fall off from least than 40 %, but with higher standard deviation than the KAOGINCSSL. Thus, we can say that both KAOGINCSSL and DWM-NB algorithms perform similarly. The SEA ensemble presents the lowest average accuracy but small standard deviation, while the WCEA instead of presenting near average mean accuracy also shows too high standard deviation, which discourages both to be used in this domain.

It is also important to notice that all the algorithms, which have used the KAOGSS as transduction algorithm, present good results, especially in the real domains. Therefore, we verify that the proposed transduction algorithm KAOGSS, not only can be used in association to the KAOGINCSSL algorithm, but also can be successfully used in other algorithms as well.

5 Conclusions

This paper has introduced a semi-supervised graph-based algorithm suitable for non-stationary streaming application, particularly when only a small portion of the acquired data presents label. Comparative results on artificial and real data sets performed on the proposed method against three well-know ensemble methods show that the proposed algorithm outperformed the compared algorithms in most of the experiments. Moreover, the results show that the present spreading label technique can be used successfully in other supervised learning algorithms to support semi-supervised classification. Future work includes testing the proposed algorithm with more data sets and comparing to other algorithms with their own spreading label method, as well as comparing the accuracy of the optimal graph as a transductive method against other transductive ones.

Acknowledgements This work is supported by the Brazilian National Research Council (CNPq) and by the São Paulo State Research Foundation (FAPESP).

References

1. Belkin M, Niyogi P (2003) Laplacian eigenmaps for dimensionality reduction and data representation. Neural Comput 15:1373–1396

2. Belkin M, Niyogi P, Sindhwani V (2006) Manifold regularization: a geometric framework for learning from labeled and unlabeled examples. J Mach Learn Res 1:1–48
3. Bertini JR Jr, Lopes A, Motta R, Zhao L (2010) Online classifier based on the optimal K-associated network. In: Proceedings of the joint conference, III international workshop on web and text intelligence (WTI'10), pp 826–835
4. Bertini JR Jr, Zhao L, Motta R, Lopes A (2011) A nonparametric classification method based on K-associated graphs. Inf Sci 181:5435–5456
5. Bornholdt S, Schuster H (eds) (2003) Handbook of graphs and networks: from the genome to the Internet, 1st edn. Wiley-VCH, Weinheim
6. Breve FA, Zhao L, Quiles M, Pedrycz W, Liu J (2011) Particle competition and cooperation in networks for semi-supervised learning. IEEE Trans Knowl Data Eng. doi:10.1109/TKDE.2011.119
7. Chapelle O, Zien A, Schölkopf B (eds) (2006) Semi-supervised learning, 1st edn. MIT Press, Cambridge
8. Chapelle O, Sindhwani V, Keerthi S (2008) Optimization techniques for semi-supervised support vector machines. J Mach Learn Res 9:203–233
9. Cormen T, Leiserson C, Rivest R, Stein C (2009) Introduction to algorithms, 3rd edn. MIT Press, Cambridge
10. Culp M, Michailidis G (2008) Graph-based semisupervised learning. IEEE Trans Pattern Anal Mach Intell 30(1):174–179
11. Ditzler G, Polikar R (2011) Semi-supervised learning in nonstationary environments. In: Proceedings of international joint conference on neural networks (IJCNN'11), San Jose, CA, USA. IEEE Press, New York, pp 2741–2748
12. Erman J, Mahanti A, Arlitt M, Cohen I, Williamson C (2007) Offline/realtime traffic classification using semi-supervised learning. Perform Eval 64:1194–1213
13. Gama J, Medas P, Castillo G, Rodrigues P (2004) Learning with drift detection. In: Proceedings of the Brazilian symposium on artificial intelligence (SBIA'04), vol 3171. Springer, Berlin, pp 286–295
14. Giraud-Carrier C (2000) A note on the utility of incremental learning. AI Commun 13(4):215–223
15. Hastie T, Tibshirani R, Friedman J (2009) The elements of statistical learning: data mining, inference and prediction, 2nd edn. Springer, Berlin
16. Hettich S, Bay S (1999) The UCI KDD archive. University of California, Irvine, School of Information and Computer Sciences. http://kdd.ics.uci.edu/
17. Kelly M, Hand D, Adams N (1999) The impact of changing populations on classifier performance. In: Proceedings of the international conference on knowledge discovery and data mining (KDD'99). ACM, New York, pp 367–371
18. Klinkenberg R, Joachims T (2000) Detecting concept drift with support vector machines. In: Proceedings of the international conference on machine learning (ICML'00). Morgan Kaufmann, San Mateo, pp 487–494
19. Kolter JZ, Maloof MA (2007) Dynamic weighted majority: an ensemble method for drifting concepts. J Mach Learn Res 8:2755–2790
20. Li P, Wu X, Hu X (2010) Mining recurring concept drift with limited labeled streaming data. In: JLMR: workshop and conference proceedings, vol 13, pp 241–252
21. Lopes AA, Bertini JR Jr, Motta R, Zhao L (2009) Classification based on the optimal k-associated network. In: Proceedings of the international conference on complex sciences: theory and applications (COMPLEX'09). Lecture notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (LNICST), vol 4. Springer, Berlin, pp 1167–1177
22. Masud M, Gao J, Khan L, Han J (2008) A practical approach to classify evolving data streams: training with limited amount of labeled data. In: Proceeding of the international conference on data mining (ICDM'08)
23. Minku L, White A, Yao X (2010) The impact of diversity on online ensemble learning in the presence of concept drift. IEEE Trans Knowl Data Eng 22:730–742
24. Narasimhamurthy A, Kuncheva L (2007) A framework for generating data to simulate changing environments. In: Proceedings of the international artificial intelligence and applications (ICAIA'07), pp 384–389
25. Quiles M, Zhao L, Alonso RL, Romero RAF (2008) Particle competition for complex network community detection. Chaos 18:033107
26. Quinlan JR (1993) C4.5 programs for machine learning, 1st edn. Morgan Kaufmann, San Mateo
27. Schaeffer S (2007) Graph clustering. Comput Sci Rev 1:27–34
28. Schlimmer J, Granger R (1986) Beyond incremental processing: tracking concept drift. In: Proceedings of the association for the advancement of artificial intelligence (AAAI'86). AAAI Press, Menlo Park, pp 502–507
29. Street N, Kim Y (2001) A streaming ensemble algorithm (SEA) for large-scale classification. In: Proc int'l conf knowledge discovery and data mining (KDD'01). ACM, New York, pp 377–382
30. Sung J, Kim D (2009) Adaptive active appearance model with incremental learning. Pattern Recognit Lett 30:359–367
31. Syed N, Liu H, Sung K (1999) Handling concept drift in incremental learning with support vector machines. In: Proceedings of the international conference on knowledge discovery and data mining (KDD'99), pp 272–276
32. von Luxburg U (2007) A tutorial on spectral clustering. Stat Comput 17:395–416
33. Wang H, Fan W, Yu P, Han J (2003) Mining concept-drifting data streams using ensemble classifiers. In: Proc international conference on knowledge discovery and data mining (KDD'03), pp 226–235
34. Widmer G, Kubat M (1996) Learning in the presence of concept drift and hidden contexts. Mach Learn 23(1):69–101
35. Yang C, Zhou J (2008) Non-stationary data sequence classification using online class priors estimation. Pattern Recognit 41:2656–2664
36. Yu Y, Guo S, Lan S, Ban T (2008) Anomaly intrusion detection for evolving data stream based on semi-supervised learning. In: Proceedings of the international conference on advances in neuro-information processing (NIPS'08), pp 571–578
37. Zhang P, Zhu X, Guo L (2009) Mining data streams with labeled and unlabeled training examples. In: Proceedings of the ninth IEEE international conference on data mining (ICDM'09). IEEE Press, New York, pp 627–636
38. Zhu X (2008) Semi-supervised learning literature survey. Tech Rep 1530, Computer-Science, University of Wisconsin-Madison
39. Zhu X (2005) Semi-supervised learning with graphs. Tech Rep Doctoral Thesis, School of Computer Science, Carnegie Mellon University