An Efficient Approach to Select Instances in Self-Training and Co-Training Semi-supervised Methods

KARLIANE MEDEIROS OVIDIO VALE¹, ARTHUR COSTA GORGÔNIO², FLAVIUS DA LUZ E GORGÔNIO¹, ANNE MAGÁLY DE PAULA CANUTO² (Member, IEEE)

¹Department of Computation and Technology, Federal University of Rio Grande do Norte, Caicó, Brazil
²Department of Informatics and Applied Mathematics, Federal University of Rio Grande do Norte, Natal, Brazil

Corresponding author: Karliane Medeiros Ovidio Vale (e-mail: karlianev@gmail.com).

This paper was supported and financed by Federal University of Rio Grande do Norte.

ABSTRACT Semi-supervised learning is a machine learning approach that integrates supervised and unsupervised learning mechanisms. In this learning, most of labels in the training set are unknown, while there is a small part of data that has known labels. The semi-supervised learning is attractive due to its potential to use labeled and unlabeled data to perform better than supervised learning. This paper consists of a study in the field of semi-supervised learning and implements changes on two well-known semi-supervised learning algorithms: self-training and co-training. In the literature, it is common to develop researches that change the structure of these algorithms, however, none of them proposes automating the labeling process of unlabeled instances, which is the main purpose of this work. In order to achieve this goal, three methods are proposed: FlexCon-G, FlexCon and FlexCon-C. The main difference among these methods is the way in which the confidence rate is calculated and the strategy used to select a label in each iteration. In order to evaluate the proposed methods’ performance, an empirical analysis is conducted, in which the performance of these methods has been evaluated on 30 datasets with different characteristics. The obtained results indicate that all three proposed methods perform better than the original self-training and co-training methods, in most analysed cases.

INDEX TERMS Artificial Intelligence, Machine Learning, Semi-supervised Learning, Self-Training semi-supervised method, Co-Training semi-supervised method.

I. INTRODUCTION

THE technological progress in recent years has greatly promoted the availability of large amounts of data. Storage and communication resources have expanded exponentially, increasing the demands for more autonomous computational tools. These tools should automatically process a considerable amount of this data, thus reducing human intervention and dependence on experts [1].

Considering this scenario, machine learning techniques have gained considerable relevance, given that, based on past experiences, they are, by themselves, capable of creating a hypothesis or function capable of solving the problem to be addressed [2].

Depending on the degree of supervision used during the training phase, Machine Learning (ML) techniques can be divided into three categories: supervised, unsupervised and semi-supervised. In these three types, the ML algorithms learn from past experience and from the implicit knowledge present in existing datasets; however, what distinguishes them is the fact that the data which such algorithms use have information that may or may not be labeled. In supervised learning, for instance, during training, the classification algorithms receive, as input, instances that contain the desired output information (labels), representing the class to which each instance belongs. On the other hand, in unsupervised learning, the classes to which the training instances belong are not previously known [3]. Finally, semi-supervised learning makes it possible to train classifiers with a small amount of labeled data and a large amount of unlabeled data [4]. This last mentioned approach for learning has become widely used in recent years [5]–[11].

Semi-supervised learning uses the previously labeled in-
To build its initial hypotheses and then combines the information obtained from those instances to label the unlabeled ones. The newly labeled instances are then inserted into the labeled dataset and these instances will serve to classify the remaining instances of the unlabeled dataset. Among the several semi-supervised learning algorithms found in the literature, the two most popular are self-training [12] and co-training [13], which will be used as the basis for the development of the proposal in this work. The main difference between these two algorithms is that self-training uses the dataset with all attributes as input, while co-training divides the input dataset into two subsets with different views.

In real-world problems, the datasets usually have few labeled instances and a lot of unlabeled instances. Then, a semi-supervised challenger deals with datasets that have these characteristics. There are many ways to solve this problem: oversampling [14], [15] and semi-supervised algorithms [9], [11], [16]. In this paper, we used semi-supervised learning.

One of the problems related to the semi-supervised learning is the automated label assignment. Considering that this process is not an easy task, especially in relation to the choice of unlabeled instances to be labeled at each stage, the authors of [17] proposed the inclusion of a parameter of confidence to guide the labeling process of the self-training algorithm. According to these authors, the general idea of using confidence in the automatic labeling process is to minimize the inclusion of noisy data, improving the overall classification accuracy.

By analyzing the above cited work, it is possible to observe a problem: the use of a confidence parameter in a static way may not exploit the full potential of a semi-supervised technique and may even increase the computational cost of the labeling process. In this way, instances with incorrect labels can be selected if the confidence parameter has a low value or instances with correct labels can be discarded if this parameter has a very high value.

As a consequence of the aforementioned problem, it can be observed that the use of a confidence parameter implies that it needs to be tuned, which is not always a trivial task. In addition, in order to obtain an effective labeling process, the confidence parameter value must be able to change during the labeling process (adaptive confidence value), given that the difficulty in selecting instances to label can change during the labeling process and a static confidence value does not capture the different levels of difficulty. Thus, the problem mentioned above was the main motivation for the development of this paper, which presents three different ways to automatically adjust the confidence parameter in the self-training method, in order to improve the efficiency of this method and to avoid a parameter tuning process. Our objective was to define ways of selecting a confidence value that was the most suitable for a specific dataset and, at the same time, adjustable during the labeling process. Later, in [11] we propose an extension of the co-training algorithm, named Co-Training with Fixed Threshold-CTFT, which uses a confidence threshold as a selection criterion of the unlabeled instances.

In this paper, we perform an exploratory investigation of our previously proposed methods (Flexcon, Flexcon-G, and FlexCon-C) [8]. The main difference between this paper and the previous one is that we extended the proposed methodology to the co-training algorithm and increased to 30 the number of datasets to investigate the performance of these methods in various contexts. Then, the main contributions of this paper can be summarized as follows:

1) As mentioned earlier, this study performs a more robust analysis of our previously proposed methods adding other algorithm and additional datasets;

2) Our methods define ways to select a confidence value that is the most suitable for a specific dataset and, at the same time, adjustable during the labeling process. It is possible because the wrapper algorithm uses a dynamic confidence threshold to include new instances in the training set at each iteration;

3) We extend the instance selection strategies for co-training algorithm, which can effectively improve the semi-supervised learning performance.

This paper is divided into seven sections as follows. Sections II and III describe the fundamental concepts and some research related to the subject of this paper, while a detailed description of the proposed method is illustrated in Section IV. Section V describes, the methodology used in the empirical analysis. Sections VI and VII present the experiments and the results provided by the empirical analysis and some discussions about them. Finally, Section VIII presents some conclusion and further works.

II. THEORETICAL REFERENCE

This section presents the main concepts related to this paper, semi-supervised learning and classification algorithms.

A. SEMI-SUPERVISED LEARNING

The semi-supervised learning mechanism handles partially labeled data in order to achieve higher classification levels [18]. Semi-supervised learning considers the set of patterns $D$ to be divided into two subsets: 1) labeled data $D_L = \{(x_i, y_i) | i = 1, \ldots, l\}$ where $x$ is the pattern, $y$ is the default label for $x$, and $l$ is the number of labeled instances; and 2) unlabeled data $D_U = \{(x_j) | j = l + 1, \ldots, l + u\}$, where $x$ is the pattern and $u$ is the number of unlabeled instances. Usually, $|D_U| \gg |D_L|$.

One of the advantages of semi-supervised learning is the potential to reduce the need for large amounts of labeled data,
particularly in domains where only a small set of labeled patterns is available. In specific domains, where there are no previously labeled datasets, it is common for an expert only has knowledge about some patterns of a given dataset, having, as a consequence, great difficulty to label instances in order to increase the training set of data [3].

In the literature, it is possible to find several algorithms that handle semi-supervised datasets, including: self-training, mix models, co-training, graph-based methods and semi-supervised support vector machines [4]. However, for the development of this work, we selected both self-training [12], considering that the work which served as the basis for this research used it, and co-training [13], because of the similarity of its classification process to self-training, differing in the use of multi-vision learning.

1) Self-training Algorithm

Self-training is probably the oldest idea on how to classify unlabeled data from a smaller subset of already labeled data. According to defined in the feature selection area, the wrapper algorithm uses a supervised method to guide their choice [19]. Then, self-training is a wrapper algorithm that repeatedly uses a supervised learning method, starting by training only the labeled data. At each step, some portion of the unlabeled instances is labeled according to the current decision function. Again, the supervised method is retrained using its own predictions as additional labeled instances [3].

In the self-training functioning, initially, a classifier is generated with a small portion of labeled data. Next, this classifier is used to classify unlabeled data. The instances that are labeled with the highest confidence index, along with their predicted labels, are included to the training set. The classifier is retrained and the procedure repeated until the unlabeled dataset is empty.

Algorithm 1 presents the sequence of steps performed to carry out this process. From this algorithm, it should be noted that the classifier uses its own predictions to learn (hence the name self-training) [4].

In [17], an extension of the self-training algorithm was implemented with the addition of a confidence parameter to be used as a threshold for including new instances in the labeled dataset. Therefore, any instance whose confidence of the prediction is greater than or equal to the minimum confidence rate for new instances to be included (threshold) will be added to the labeled dataset.

2) Co-training Algorithm

The co-training algorithm, initially proposed by [13], is similar to self-training, given that it increases the set of labeled data by iteratively classifying the set of unlabeled data and moving the most reliable prediction instances to the set of labeled data. However, differently from self-training, two complementary classifiers are simultaneously generated, fed with two different views of the set of attributes $A = \{a_1, \ldots, a_n\}$.

![Algorithm 1: Self-Training Algorithm](image)

In this approach, these two complementary classifiers proceed as following: after generating the two views of the data, the prediction of the first classifier is used to increase the labeled dataset available for the second classifier and vice versa [20]. In other words, the prediction of one classifier is
presented to the other (and vice versa) and their outputs are combined.

At the beginning two views are created, then two supervised classifiers \(C_1\) and \(C_2\) are generated based on the labeled data of each of the two views. The next step is to classify the unlabeled data for each of the views, using classifier \(C_1\) to label View 1 and \(C_2\) to label View 2. Afterwards, the instances with highest confidence in the prediction of the classifier \(C_2\) will be added to the labeled dataset in View 1. In the same way, View 2 will receive the instances with the highest confidence in the prediction of the classifier \(C_1\). This process is repeated until the set of unlabeled data is empty. Algorithm 2 corresponds to the series of steps performed to carry out this process.

**Algorithm 2: Co-training Algorithm**

**Input:** labeled data \(L = \{ (x_i, y_i) | i = 1, \ldots, l \}\),
unlabeled data \(U = \{ (x_j) | j = l + 1, \ldots, l + u \}\).
Each instance has two views \(x_i = [x_i^{(1)}, x_i^{(2)}]\).
Let \(A\) the set of the attributes \(A = \{a_1, a_2, \ldots, a_n\}\),
where \(A^{(1)}\) and \(A^{(2)}\) represents the Views 1 and 2, respectively. In addition, \(A^{(1)} \bigcap A^{(2)} = \emptyset\).
Initially, we have all of the training instances as
\[
D^{(1)} = \{ (x_i^{(1)}, y_i), \ldots, (x_l^{(1)}, y_l) \} \quad \text{and} \quad \quad D^{(2)} = \{ (x_i^{(2)}, y_i), \ldots, (x_l^{(2)}, y_l) \}
\]
and the unlabeled instances as
\[
D^{(1)} = \{ x_{l+1}^{(1)}, \ldots, x_{l+u}^{(1)} \} \quad \text{and} \quad \quad D^{(2)} = \{ x_{l+1}^{(2)}, \ldots, x_{l+u}^{(2)} \}
\]
where,
\[
\{ D_U \} = D^{(1)} \bigcup D^{(2)} \\
\{ D_U \} = D^{(1)} \bigcup D^{(2)}
\]

1: repeat
2: Generate classifiers \(C^{(1)}\) and \(C^{(2)}\) based on training data \(D^{(1)}\) and \(D^{(2)}\), respectively.
3: Classify unlabeled data \(D^{(1)}\) and \(D^{(2)}\) using classifiers \(C^{(1)}\) and \(C^{(2)}\), respectively.
4: Add to the set \(D^{(2)}\) the instances classified by \(C^{(1)}\) with a prediction confidence equal to the maximum confidence value achieved.
5: Add to the set \(D^{(1)}\) the instances classified by \(C^{(2)}\) with a prediction confidence equal to the maximum confidence value achieved.
6: Remove these instances from the unlabeled dataset.
7: until \(D_U = \emptyset\)

**Output:** labeled data

### B. CLASSIFICATION STRUCTURES

As it can be observed in the previous section, the semi-supervised methods apply a classification algorithm in order to label the unlabeled instances. Therefore, in this section, a brief description of classification structures is presented.

In the context of machine learning, a classifier is a technique that formulates a hypothesis based on a sample of data [21]. In other words, a classification system creates a model using the customer’s existing data in a useful way to calculate the risk for a new application and then decides whether or not to approve the requested credit [22].

Several algorithms can be used in classification systems, the main differences refer to the learning strategy, representation language, and the amount of knowledge previously used [23]. The classifiers used in this work are namely: Naïve Bayes [24], decision tree [25], Ripper [26] and \(k\)-NN [27].

The increased complexity and wide applicability of classification systems has led to exploring many approaches and methodologies. Nevertheless, there is a perception that no classifier is considered completely satisfactory for a particular task; therefore, the idea of combining different methods to improve performance has emerged as a very promising possibility [28]. This combination is called classifier ensembles, also known as multi-classifier systems.

In classification tasks, an ensemble includes several sub-models called base classifiers, which are usually obtained by training a basic learning algorithm (decision tree, neural network, \(k\) nearest neighbors, among others). The ensembles can be built based on the same learning algorithm, producing homogeneous ensembles, or using different algorithms and generating heterogeneous ensembles [29].

The proposition of classifier ensembles is to create and combine several inductive models for the same domain, obtaining better prediction quality [30]. After generating the base classifier set, the next step is the choice for the methods to combine their outputs. There is a vast number of methods for combining classifiers in the literature.

In this work, two well-known combination methods will be used (sum and voting). The choice for these methods was made due to the use of information from all classifiers. Sum is one of the simplest and most widely used combination methods. In this method, once the base classifiers have generated their outputs (degree of membership to each of the classes) for an instance, all the outputs of each classifier, for each of the classes, are added up and the winning class is the one that has the highest absolute value. The voting method, often used to combine classifiers, performs the combination by voting on the results of each classifier when a new instance is presented.

### III. RELATED WORK

In the literature, there are several works in the field of semi-supervised learning [31]–[33]. Some researches address the self-training and co-training algorithms, which seek to define ways to evaluate these algorithms [34] or to solve problems [16], [35]–[38], others to make changes in the structure of these algorithms, creating extensions or new algorithms [6], [7], [9], [17], [39]–[41], in addition to their use in different applications [42]–[44]. The research developed in this article fits into the group of researches which change the structure of these algorithms.

In [31], a novel neural embedding matching (NEM) method was proposed to tackle domain adaptation by en-
forcing consistent class-wise cross-domain instance distributions in the embedding space. According to the authors, the proposed method is a progressive learning strategy, which can improve the semi-supervised learning effectively. In [32], the authors proposed a multiple kernel active learning algorithm, which incorporates distribution matching with multiple kernel learning as a group lasso into uncertainty. In [33], the authors proposed two novel semi-supervised frameworks which combine secondary screening algorithm and semi-supervised learning for guaranteeing the diversity of samples. They are named the syncritic one-fold secondary screening algorithm and semi-supervised learning framework (OFSS-SL) and syncritic multiple secondary screening algorithms and multiple-verification semi-supervised learning framework (MSS-MVSL).

In [34], the objective is to use a decision tree as a basic learner for the self-training algorithm. The research clarified that by improving the probability estimation of the decision tree, self-training improves its performance. In [35], a semi-supervised approach was proposed which adapts active learning to the co-training algorithm as a way classify hyperspectral images, automatically selecting new training samples from unlabeled pixels. The effectiveness of the proposed approach is validated by using a probabilistic support vector machine classifier.

Among the studies that solve problems in semi-supervised methods, in [36], the objective is to design a semi-supervised learning model for the NER (Indonesian Named Entity Recognition) system. NER aims to identify and classify an entity based on its context, however few instances have a label. Therefore, co-training, as a semi-supervised learning model, was used to handle unlabeled data in the NER learning process and produce new labeled data that can be applied to improve a new NER classification system. Additionally, in [37], a new approach is proposed for classifying students in an academic credit system, combining transfer of learning and co-training. The resulting model can effectively predict the study status of a student enrolled in a given educational program, which is done by means of a classification model enhanced by transfer learning and co-training techniques operating on educational data from another program.

In [38], the authors proposed a semi-supervised framework that use an ensemble in order to identify wrongly classified instances. This approach uses these instances, also Self-training and F-measure estimations, to improve an ensemble. In addition, this framework is based on error detection and unsupervised accuracy estimation, which are combined to build an ensemble to classify instances. The obtained results perform very accurate estimations for error and F-measure. In [16], a semi-supervised approach was proposed for recommendation attack detection, based on Co-forest algorithm, that presents two main aspects: i) a feature extraction method; and ii) a detection method. The feature extraction method uses a window to extract features to be used to train the detection method. The detection step denotes whether it is a genuine or an attack profile. In order to determine the type of profile, an ensemble must be fitted using the Co-forest approach.

As previously mentioned, it is possible to find in the literature different studies which make changes in the structure of the self-training and co-training algorithms in order to create new algorithms or extensions [6], [7], [9], [17], [39]–[41]. In [6], for instance, the authors developed a new self-training style algorithm which explores a multiple hypothesis to optimize the self-labeling process. This process uses a graph-based transductive method to generate reliable predictions. In contrast to the standard self-training, the proposed algorithm uses labeled and unlabeled data as a whole to label and select unlabeled instances to increase the training set. According to the author, the proposed algorithm has several interesting properties, among them: it can generate more reliable labels in unlabeled data and has a strong tolerance to noise in the training set.

Following the previous context, in [7], the authors proposed a framework for semi-supervised self-training classification, whose structure of the data space distribution (spherical or non-spherical) was integrated into the self-training process. This framework consists of two main parts, one is discovering the real structure of the entire data space, researching and locating spikes in data density; the other is the integration of the actual structure of the entire data space into the self-training process to iteratively train a classifier. Continuing with some more examples of studies that changes the structure of the self-training algorithm, in [17], the authors proposed four semi-supervised methods, based on self-training, which can be applied to multi-label classification problems. The main idea of these methods is to minimize the randomness in all the instances are chosen in the labeling process. In this way, the main objective of the work is to use a confidence parameter in the automatic data labeling process, aiming to minimize the inclusion of noise, improving, in general, the accuracy of the classification. Therefore, only instances whose classifier output labels have confidence values above a confidence threshold are taken into account. In this sense, this confidence threshold (value between 0 and 1), was used to control the automatic label assignment in the semi-supervised learning process.

Among the studies that create new algorithms or extensions for co-training, in [40], the Deep Co-training method was created, which is a method based on deep learning, inspired by the structure of co-training. Deep Co-training trains several deep neural networks to be used with the different views required for co-training to work and explores contradictory examples to encourage differences between the views in order to prevent the networks from collapsing with each other. In [9] a method was proposed, called multi-co-training, whose objective is to improve the performance of document classification. Documents are transformed using three methods of document representation to increase the variety of attribute sets for classification.

In [41], a semi-supervised learning algorithm is introduced, combining co-training with a support vector machine...
classification algorithm (SVM – Support Vector Machine). By using an interactive learning procedure, the new labeled dataset can be determined based on unlabeled datasets, training two SVM classifiers.

Finally, some studies use the self-training method in different application domains [42], [44]. For example, in [44], the authors proposed a semi-supervised learning model for the sentiment analysis problem called interpolative self-training. As the name indicates, this model is an extension of the self-training algorithm, and its main difference is the concatenation of training and test data. In [42], the authors presented a new version for self-training used in the recognition of manuscripts which is based on neural networks.

IV. THE PROPOSED METHODS

As discussed earlier, the main objective of this work is the proposal to extend the semi-supervised algorithms (self-training and co-training), aiming to improve their efficiency. Therefore, the proposed approach consists of including a confidence parameter, adjustable to each iteration, which can be used as a threshold for the inclusion of new instances in the labeled data set.

A. CONFIDENCE ADJUSTMENT METHODS

As mentioned earlier, in traditional semi-supervised learning algorithms, a single classifier is iteratively trained with a growing set of labeled data. It starts with a small portion of labeled instances, adding new instances to each iteration. However, it is observed that the automatic label assignment process is a difficult task and the main issue is related to the choice of unlabeled instances to be labeled.

In [17] a confidence parameter was included in the labeling process. In the extension of the self-training presented in the cited work, unlabeled instances whose confidence rate of the prediction is greater than the confidence threshold are added to the training set, along with their predicted labels. However, in the above cited algorithm, the authors used a static value for the minimum confidence rate, which may not take advantage of the full potential of a semi-supervised method. Hence, if the parameter has a low value, it is possible for instances with incorrect labels to be selected. If the parameter is too high, on the other hand, it is possible that instances with correct labels are discarded.

In this context, the objective of this work is to provide flexibility in the confidence threshold to include new instances in the labeled dataset, allowing the labeling process to be carried out according to a real practical scenario. In this work, several methods will be proposed to calculate, at each iteration, the confidence threshold for the inclusion of new unlabeled instances, among them: FlexCon-G, FlexCon and FlexCon-C. These three methods will be described in the next subsections.

1) FlexCon-G Method

The FlexCon-G method (Flexible Confidence with Graduation) adjusts the confidence value gradually, with the user initially setting a high confidence threshold (value close to 100%) to be used in the first iteration. Then, a fixed rate \(d\) is defined and, in each iteration, the confidence threshold is decreased by this rate \(d\). The formula for calculating confidence in the current iteration, \(conf(t_{i+1})\), is described in Equation (1):

\[
conf(t_{i+1}) = conf(t_i) - d
\]

where \(conf(t_i)\) is the confidence value in the previous iteration and \(d\) is the rate in which the confidence threshold decreases.

The purpose of this method is to start with a high threshold (confidence value) and to decrease it gradually throughout the labeling process. In the initial phase, the classification algorithms have a small set of labeled data \(D_L\) and a restrictive threshold is defined. As the labeling process progresses, the labeled dataset \(D_L\) increases and a lower confidence limit is then used.

In addition, each instance included in the labeled dataset has its label defined by the output of the used classifier, in the same way as the semi-supervised learning methods in their original form do.

2) FlexCon Method

In the method for adjusting confidence in a flexible way, called FlexCon (Flexible Confidence), the equation that calculates the new confidence rate is based on three different aspects:

1) The confidence rate of the previous iteration;
2) The accuracy of a classifier which uses the labeled instances in the previous iteration as a training set (with the labels predicted by the classifier) and the dataset initially labeled as a test set;
3) The percentage of instances labeled in the previous iteration.

Based on these aspects, Equation (2) calculates the arithmetic mean between the three parameters mentioned, being then defined as:

\[
conf(t_{i+1}) = \frac{1}{3}(conf(t_i) + \frac{1}{|L_i|} \sum_{j=1}^{|L_i|} prec(s_j)) + \frac{|L_i|}{|D_U|},
\]

where \(conf(t_{i+1})\) is the confidence value in the current iteration, \(conf(t_i)\) is the confidence value in the previous iteration, \(|L_i|\) is the number of instances labeled in the previous iteration \(t\), \(s_j\) is each of the patterns in the set \(L_i\); \(prec(s_j)\) is the precision of the \(s_j\) pattern in time \(t_i\); \(|D_U|\) is the number of instances of the unlabeled dataset.

This method is aimed at adjusting the confidence threshold based on its value in the previous iteration, as well as in the precision \(prec\) and coverage \(|L_i|/|D_U|\) of the classifiers generated with a basis on the information from the previous iteration. Figure 2 presents an example of calculating the confidence threshold using FlexCon. Assuming that the second
FlexCon iteration will be performed, it is necessary to know about the first iteration (box on the left side of the figure). The box on the left displays three pieces of information:

1) The value of the confidence threshold on the first iteration \( (\text{conf}(1)) \);
2) The labeled dataset, with two columns: instance \((x_1, x_2, \ldots, x_5)\) and label (A or B);
3) The classifier prediction for the still unlabeled dataset, with the following columns: instance \((x_6, x_7, \ldots, x_{11})\), predicted label (A or B) and the confidence in the classifier’s prediction, which must lie in the \([0, 1]\) interval and it represents the certainty which the classifier achieved that the instance belongs to the predicted class.

Still in the left box, the instances of the unlabeled dataset that are shaded \((x_6, x_7, \text{ and } x_8)\) are those whose confidence values in the prediction are greater than or equal to the confidence threshold \( (\text{conf}(1)) \). In other words, the instances that will be included in the labeled dataset. The boxes on the right show the calculation of the new confidence threshold value that will be used in the second iteration, according to Equation 2.

As it can be seen, the labeling process has two steps: As it can be seen, the labeling process has two steps: instance selection (from all classes) and label assignment. The use of the confidence rate must occur in the selection of the instances to be labeled. The labeling itself can be done by a classification method. In this work, we will use two classification structures, individual classifiers and classifier ensembles. For the case of using ensembles, one way to further improve the performance of the labeling process is to use an ensemble that is composed of classifiers built in all iterations carried out up to the current iteration. For example, in the fifth iteration, the classifier ensemble is composed of four individual classifiers, built in the previous four iterations. These individual classifiers are combined in two ways, to lead to two different versions of this method: (1) simple sum (FlexCon \( s \) version), the partial confidence values are aggregated using the individual classifier predict, and it is selected the label which has the highest aggregated value; and (2) majority vote (FlexCon \( v \) version), electing the most voted label among those presented by the individual classifiers. Majority vote and sum methods have been selected in this work since they are robust and widely used in the literature to combine classifier results. However, there are several other approaches for combination methods which are more efficient, such as alpha integration for an optimal fusion of individual classifiers [45], stacking [46], among others. The use of more efficient combination methods is the subject of an ongoing investigation.

3) FlexCon-C Method

In the method for adjusting confidence on a flexible way using classifiers - FlexCon-C (Flexible Confidence Classifier) - the confidence threshold is adjusted by increasing or decreasing its value based on a change rate \( (\text{cr}) \). In this way, the confidence threshold is decreased by the change rate \( (\text{cr}) \), when the classifier precision \( (\text{acc}) \) is greater than the minimum acceptable precision \( (\text{mp}) \). Otherwise, the confidence threshold is increased by the change rate \( \text{cr} \), if the classifier’s accuracy is less than the minimum acceptable precision. When the classifier precision \( (\text{acc}) \) is in an acceptable minimum precision range \( (\text{mp}) \), then the confidence threshold will not be changed and the confidence rate for the current iteration remains the same as in the previous iteration. In all cases, it is necessary to consider, as a safety margin, an acceptable variation \( (e) \) of the minimum precision.

Equation (3) presents the calculation of the confidence in the current iteration, \( \text{conf}(t_{i+1}) \), using FlexCon-C:

\[
\text{conf}(t_{i+1}) = \begin{cases} 
\text{conf}(t_i) - \text{cr}, & \text{if } \text{acc} \geq \text{mp} + e, \\
\text{conf}(t_i) + \text{cr}, & \text{if } \text{mp} - e < \text{acc} < \text{mp} + e, \\
\text{conf}(t_i), & \text{if } \text{acc} \leq \text{mp} - e,
\end{cases}
\]

where \( \text{conf}(t_{i+1}) \) is the confidence value during the current iteration; \( \text{mp} \) is the minimum acceptable precision; \( \text{cr} \) is the change rate; and \( \text{acc} \) is the classifier precision obtained in the previous iteration. Finally, \( e \) represents an acceptable variation in precision.

Figure 3 shows an example of calculating the confidence threshold using FlexCon-C. In order to do so, the minimum precision value is defined as the accuracy obtained from the initially labeled dataset, which is used for training and testing (box on the upper left side of figure 3). Assuming that the second iteration of FlexCon-C will be performed, it is necessary to know the information related to the first iteration, which is shown in the box on the lower left side:

1) The confidence threshold value in the first iteration \( (\text{conf}(1)) \);
2) The labeled dataset, with two columns: instance \((x_1, x_2, \ldots, x_5)\) and label (A or B);
3) The classifier prediction for the still unlabeled dataset, with the following columns: instance \((x_6, x_7, \ldots, x_{11})\), predicted label (A or B) and the confidence in the classifier’s prediction, which must be between \([0, 1]\) and represents the certainty which the classifier achieved that the instance belongs to the predicted class.
It is possible to observe that these images differ from original self-training and co-training, respectively, once it includes the dashed blocks.

A. EXPERIMENTAL METHODOLOGY USING SELF-TRAINING

In the methodology based on self-training (Figure 4), a supervised classifier is initially generated based on the set of labeled data, which will be used to classify the unlabeled data. Then, the new confidence threshold value which is used in the selection of new instances to be labeled will be calculated. In the next step, the instances whose confidence value for the prediction is greater than or equal to the confidence threshold will be selected, which are labeled according to different strategies. Finally, the process will be restarted using the new labeled dataset until the unlabeled dataset is empty.

The Algorithm 3 shows the step by step of the new version of self-training implemented in the development of this work, having the lines highlighted in black indicating the main differences in relation to original self-training (Algorithm 1).

Algorithm 3: Algorithm for Self-training with Confidence Adjusting

Input: labeled data \( \{D_L\} \), unlabeled data \( \{D_U\} \)
Initially, we have \( \{D_L\} = \{(x_l, y_l)|l = 1 \ldots l\} \) and \( \{D_U\} = \{(x_j)|j = l + 1 \ldots l + u\} \)

1: repeat
2: Train classifier \( C \) based on \( \{D_L\} \) using semi-supervised learning;
3: Apply \( C \) on the instances in \( \{D_U\} \);
4: Calculate a new value for the confidence rate
5: Remove a subset \( S = \{s_1, s_2, \ldots, s_n\} \) from \( \{D_U\} \), so that the confidence rate in \( C(x) \) is greater than or equal to the minimum confidence rate for new instances to be included;
6: Use different strategies to choose the label for every instance in subset \( S \);
7: Add \( \{(x, C(x)|x \in S\} \) to set \( \{D_L\} \);
8: until \( \{D_U\} = \emptyset \)

Output: labeled data

The self-training presented in [17] is different from the proposal of this work in the following ways:

V. EXPERIMENTAL METHODOLOGY

The experimental methodology used to apply the methods proposed in this work is similar to the process of labeling used by the self-training and co-training algorithms. Figure 4 represents how the data should be trained and labeled, according to the self-training process, using the methods explained above. Figure 5 corresponds to the co-training process. The dashed blocks represents the main difference between these process and the original self-training and co-training.
• The extension of the self-training proposed in [17] does not change the confidence threshold to include new instances, whereas this proposal allows for variation of the said value for each iteration (see Algorithm 3 Line 4);
• In the extension of self-training proposed in [17], the label provided by the classifier is directly assigned to an unlabeled instance when it is moved to the labeled dataset. In this work, different strategies are proposed to define the correct label, among them: classifier ensembles (see Algorithm 3, Line 6).

B. EXPERIMENTAL METHODOLOGY USING CO-TRAINING

The methodology proposed in this experiment, based on co-training (Figure 5), starts by creating two different views (1 and 2) of the dataset. Then, two supervised classifiers are generated based on the labeled datasets ($G_1$: View 1 and $G_2$: View 2), which will be used to classify the unlabeled data. Right after this, the new confidence threshold value is calculated for each of the classifiers separately, which are used in the selection of new instances to be labeled. In the next step, the instances whose confidence value in the prediction is greater than or equal to this threshold defined for the classifier by which the instance was classified will be selected. These same instances are, then, labeled using different strategies. However, the most reliable instances predicted by the classifiers $C_2$ and $C_1$ will be added to the labeled dataset of Views 1 and 2, respectively. Finally, the process is restarted using the new labeled datasets until both unlabeled datasets are empty.

![FIGURE 5. Experimental Methodology based on Co-training.](image)

In this work, an extension of the co-training algorithm was developed following the same methodology as [17], i.e., using a static confidence threshold for selecting instances in the labeling process. The implementation of this new extension was necessary to make it possible to compare the performance of co-training using a fixed confidence threshold and the proposed methods whose threshold is flexible. A new extension of the co-training algorithm is presented in Algorithm 4. This algorithm uses a fixed threshold to select new training instances. As a co-training algorithm, this one uses two views with distinct attributes. The main difference between this method and original semi-supervised co-training is how the confidence parameter (threshold) is measured (Lines 4-5 of Algorithm 4). The semi-supervised co-training algorithm proposed by [13] selects and uses the highest confidence value as the threshold to select instances with the same value. On the other hand, our proposal uses a prefixed confidence parameter to select all instances whose prediction confidence rate is greater than or equal to this threshold. The lines in black represent the differences between this algorithm and the original co-training (Algorithm 2).

Algorithm 4: Algorithm for Co-training Using Static Threshold

**Input**: labeled data $\{D_L\} = \{(x_i, y_i)|i = 1, \ldots, l\}$, unlabeled data $\{D_U\} = \{(x_j)|j = l + 1, \ldots, l + u\}$. Each instance has two views $x_i = [x_i^{(1)}, x_i^{(2)}]$.
Let $\mathcal{A}$ the set of the attributes $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$, where $A^{(1)}$ and $A^{(2)}$ represents the Views 1 and 2, respectively. In addition, $A^{(1)} \cap A^{(2)} = \emptyset$.

Initially, we have all of the training instances as $D^{(1)} = \{(x_i^{(1)}, y_i), \ldots, (x_{l+u}^{(1)}, y_i)\}$ and $D^{(2)} = \{(x_i^{(2)}, y_i), \ldots, (x_{l+u}^{(2)}, y_i)\}$ and the unlabeled instances as $D^{(1)} = \{(x_i^{(1)+1}), \ldots, (x_{l+u}^{(1)+1})\}$ and $D^{(2)} = \{(x_i^{(2)+1}), \ldots, (x_{l+u}^{(2)+1})\}$ where,
$$\{D_L\} = D^{(1)} \cup D^{(2)}$$
$$\{D_U\} = D^{(1)} \cup D^{(2)}$$

1: repeat
2: Generate classifiers $C^{(1)}$ and $C^{(2)}$ based on training data $D^{(1)}_L$ and $D^{(2)}_L$, respectively.
3: Classify unlabeled data $D^{(1)}_U$ and $D^{(2)}_U$ using classifiers $C^{(1)}$ and $C^{(2)}$, respectively.
4: Add to set $D^{(2)}_L$ the instances classified by $C^{(1)}$, whose confidence rate of the prediction is greater than or equal to the minimum confidence rate for new instances to be included.
5: Add to set $D^{(2)}_L$ the instances classified by $C^{(2)}$, whose confidence rate of the prediction is greater than or equal to the minimum confidence rate for new instances to be included.
6: Remove these instances from the unlabeled dataset.
7: until $\{D_U\} = \emptyset$

**Output**: labeled data

The methods proposed in this work, FlexCon-G, FlexCon and FlexCon-C, were also applied to the extension of co-training which uses a static threshold, previously presented in Algorithm 4. Algorithm 5 shows series of steps of the new version of the co-training, with confidence adjustment, to be implemented in the development of this work and the lines marked with black indicate the main differences of Algorithm 4.

The co-training algorithm using a static confidence threshold (described in Algorithm 4 in this Section) differs from the proposal of this work in the following aspects:
• The extension of co-training using a fixed threshold (Algorithm 4) does not change the confidence threshold to
include new instances, whereas this proposal allows the variation of this value for each iteration (see Algorithm 5, Line 10);

- In the extension of the co-training using a static threshold (Algorithm 4), the label provided by the classifier is directly assigned to an unlabeled instance when it is moved to the labeled data set. In this work, different strategies are proposed to define the correct label, among them: classifier ensembles (see Algorithm 5, Line 13).

**Algorithm 5:** Algorithm for Co-training with Confidence Adjustment

**Input:** labeled data \( D_L = \{(x_i, y_i) | i = 1, \ldots, l\} \),
unlabeled data \( D_U = \{(x_j) | j = l + 1, \ldots, l + u\} \).

Each instance has two views \( x_i = [x_i^{(1)}, x_i^{(2)}] \).

Let \( A \) the set of the attributes \( A = \{a_1, a_2, \ldots, a_n\} \), where \( A^{(1)} \) and \( A^{(2)} \) represents the Views 1 and 2, respectively. In addition, \( A^{(1)} \cap A^{(2)} = \emptyset \).

Initially, we have all of the training instances as:

\[
D_U^{(1)} = \{(x_1^{(1)}, y_1), \ldots, (x_l^{(1)}, y_l)\}
\]

and the unlabeled instances as:

\[
D_U^{(2)} = \{(x_1^{(2)}, y_1), \ldots, (x_l^{(2)}, y_l)\}
\]

where,
\[
D_L = D_U^{(1)} \cup D_U^{(2)}
\]

\[
D_U = D_U^{(1)} \cup D_U^{(2)}
\]

**repeat**

1. Generate classifiers \( C^{(1)} \) and \( C^{(2)} \) based on the training data \( D_U^{(1)} \) and \( D_U^{(2)} \), respectively.
2. Classify the unlabeled data \( D_U^{(1)} \) and \( D_U^{(2)} \) using classifiers \( C^{(1)} \) and \( C^{(2)} \), respectively.
3. Calculate the new value for the confidence threshold.
4. Add to set \( D_U^{(2)} \) the instances classified by \( C^{(1)} \), whose confidence rate of the prediction is greater than or equal to the minimum confidence rate for new instances to be included.
5. Add to set \( D_U^{(1)} \) the instances classified by \( C^{(2)} \), whose confidence rate of the prediction is greater than or equal to the minimum confidence rate for new instances to be included.
6. Use different strategies to choose the label for every new instance included in the labeled datasets \( D_U^{(1)} \) and \( D_U^{(2)} \).
7. Remove these instances from the unlabeled dataset.
8. until \( \{D_U\} \neq \emptyset \)

**Output:** labeled data

**VI. EXPERIMENTS**

**A. DATASETS**

To validate the feasibility of the proposed confidence adjustment methods, an empirical analysis was performed. In this analysis, 30 different sets of classification data were used. The datasets used in these experiments were obtained from several repositories: UCI Machine Learning (UCI) [47], Knowledge Extraction based on Evolutionary Learning (KEEL) [48], Kaggle Datasets [49] and GitHub [50]. Table 1 briefly describes the datasets used, in terms of the number of instances (#Inst), attributes (#Att) and classes (#Class) in each dataset. In addition, it indicates the data type (Type), either integer (I) and/or categorical (C) and/or real (R).

| Dataset         | #Inst | #Att | #Class | Type |
|-----------------|-------|------|--------|------|
| Balance Scale   | 625   | 5    | 3      | C    |
| BTSC            | 748   | 5    | 2      | C    |
| Bupa            | 345   | 7    | 2      | C,R  |
| Car Evaluation  | 1728  | 7    | 4      | C    |
| Cnae-9          | 1080  | 857  | 9      | I    |
| Connectionist Bench | 208 | 60   | 2      | R    |
| Hill Valley With Noise | 606 | 101  | 2      | R    |
| Image Segmentation | 2310 | 19   | 7      | R    |
| Indian Liver Patient | 583 | 10   | 2      | LR   |
| Iris            | 150   | 4    | 3      | R    |
| KR vs KP        | 3196  | 37   | 2      | C    |
| Leukemia        | 100   | 50   | 2      | R    |
| Manomographic Mass | 961 | 6    | 2      | I    |
| Multiple Features | 2000 | 649  | 10     | R    |
| Mushroom        | 8124  | 22   | 2      | C    |
| Musk            | 6598  | 168  | 2      | I    |
| Ozone Level Detection | 2536 | 73   | 2      | R    |
| Pen Digits      | 10992 | 16   | 10     | I    |
| Phishing Website | 2456 | 30   | 3      | I    |
| Pima            | 768   | 9    | 2      | LR   |
| Planning Relax  | 182   | 13   | 2      | R    |
| Seeds           | 210   | 7    | 3      | R    |
| Semeion         | 1593  | 256  | 10     | I    |
| Solar Flare     | 1389  | 10   | 3      | C    |
| SPECTF Heart    | 267   | 44   | 2      | I    |
| Tic-Tac-Toe Endgame | 958 | 9    | 2      | C    |
| Twnomir         | 7400  | 21   | 2      | R    |
| Vehicle         | 946   | 18   | 4      | I    |
| Waveform        | 5000  | 40   | 3      | R    |
| Wilt            | 4839  | 6    | 2      | R    |

1 Blood Transfusion Service Center.
2 King-Rook vs King-Pawn.
3 Pen-based recognition of handwritten digits.

**B. DATASET SPLITTING**

Figure 6 shows, as an example, how the datasets were organized before training. It is common to have a sample of data for training and another independent sample, with different data, for testing. As long as both samples are representative, the error rate in the test suite will give a good indication of performance [51]. Thus, in the experiments of this work, each dataset was divided into two sets:

1. A training set, with 90% of the instances;
2. A test set, with the remaining instances. (10%).

From each dataset, 10 repetitions were obtained by a different training/test splitting (cross-validation). Given that all datasets were originally labeled, it was possible to perform, in a stratified manner, 5 different configurations using 55%, 10%, 15%, 20% and 25% of the data initially labeled.
C. CONFIGURATIONS

After pre-processing the data, the training/test procedure starts. In this analysis, four classification algorithms well known in the literature were applied: Naive Bayes (NB), decision tree (DT), rule based classification algorithm (Ripper) and $k$ closest neighbors ($k$-NN). As explained earlier, such algorithms were chosen because of their popularity and use in machine learning studies. For all algorithms, Weka implementations available in the R language were used. Details on the operation of the classifiers and their implementation in the R language are described in [30] and [52].

Additionally, at the beginning of the labeling process for self-training and co-training algorithms, it is worth mentioning that the set of labeled data has a small number of instances. Therefore, it is necessary to assign labels more cautiously, which is precisely why a high threshold will be used. However, there is no way to guarantee that the same value can be considered high for all datasets, since each dataset has an intrinsic degree of difficulty. In view of the above, the initial value of the confidence threshold becomes difficult to estimate and requires further investigation. However, the investigation regarding the most suitable value for this threshold is not the objective of this work and, for this reason, a sample test was carried out using only the 90% of the labeled data. The choice of the labeled dataset is made randomly, but in a stratified manner, respecting the same proportion of classes as the initially labeled dataset.

A. PERFORMANCE ANALYSIS

This section will present the analysis of results of the experiments which evaluate the performance of the methods proposed in this work. In order to facilitate the discussion, the results are evaluated taking into account the average accuracy of the algorithms for all datasets, as well as the standard deviation of these measures. The tables showing the results are evaluated taking into account the average accuracy of the algorithms for all datasets, as well as the standard deviation of these measures. The tables showing the results are evaluated taking into account the average accuracy of the algorithms for all datasets, as well as the standard deviation of these measures. The tables showing the results are evaluated taking into account the average accuracy of the algorithms for all datasets, as well as the standard deviation of these measures.

The original self-training and co-training methods were called original ST and original CT, respectively. The methods which do not vary the value of confidence for the inclusion of new instances throughout the process, when applied to self-training and co-training are called, respectively, ST Fixed Threshold (STFT) and CT Fixed Threshold (CTFT). While for the methods proposed in this work, the same nomencla-
ture was used for both self-training and co-training: FlexCon-G, FlexCon(s), FlexCon(v), FlexConC1(s), FlexConC1(v) and FlexCon-C2. In addition, results whose classification accuracy is superior to the original self-training and co-training, are highlighted in bold, while the cells shaded in yellow represent the proposed methods whose accuracy was better than the method using fixed threshold.

The following sections present the analysis of the performance of each method when using self-training and co-training semi-supervised learning algorithms, respectively.

### 1) Performance Analysis with Self-training

Table 2 shows the arithmetic mean of the accuracy and standard deviation of each method, using Naive Bayes, decision tree, Ripper and k-NN as classification algorithms.

**TABLE 2.** Average Accuracy of all methods using Naive Bayes, decision tree, Ripper and k-NN classifiers with self-training

| Method          | % instances initially labeled | 5%          | 10%          | 15%          | 20%          | 25%          |
|-----------------|------------------------------|-------------|-------------|-------------|-------------|-------------|
| Naive Bayes     |                              |             |             |             |             |             |
| Self-trained    | 87.77 ± 1.49                 | 86.86 ± 1.23 | 87.49 ± 1.26 | 87.01 ± 1.24 | 86.71 ± 1.03 |
| Decision Tree   | 68.58 ± 1.72                 | 76.33 ± 1.79 | 71.13 ± 1.89 | 70.81 ± 1.85 | 70.11 ± 1.03 |
| Ripper          | 69.20 ± 1.74                 | 69.97 ± 1.85 | 70.48 ± 1.85 | 70.72 ± 1.85 | 70.30 ± 1.03 |
| k-NN            | 70.66 ± 1.97                 | 70.89 ± 1.78 | 71.59 ± 1.42 | 71.76 ± 1.75 | 71.58 ± 1.17 |
| G               | 76.91 ± 1.31                 | 76.71 ± 1.49 | 76.92 ± 1.19 | 74.19 ± 1.92 | 71.64 ± 1.85 |
| FlexCon(s)      | 76.58 ± 1.48                 | 76.91 ± 1.39 | 76.89 ± 1.10 | 71.84 ± 1.19 | 71.92 ± 1.84 |
| FlexCon-C1(s)   | 69.83 ± 1.74                 | 70.67 ± 1.20 | 71.49 ± 1.58 | 71.54 ± 1.12 | 71.55 ± 1.74 |
| FlexCon-C1(v)   | 70.94 ± 1.80                 | 75.50 ± 1.34 | 75.39 ± 1.39 | 76.85 ± 1.13 | 76.12 ± 1.03 |
| FlexCon-C2      | 70.21 ± 1.43                 | 70.76 ± 1.25 | 70.84 ± 1.39 | 79.07 ± 1.46 | 79.16 ± 1.34 |
| FlexCon-C1(v)   | 70.50 ± 1.18                 | 70.32 ± 1.33 | 70.85 ± 1.67 | 70.57 ± 1.39 | 80.03 ± 1.23 |
| FlexCon-C2      | 70.27 ± 1.70                 | 71.27 ± 1.40 | 71.85 ± 1.36 | 71.16 ± 1.19 | 70.05 ± 1.03 |
| G               | 79.06 ± 1.94                 | 79.78 ± 1.44 | 79.71 ± 1.07 | 78.30 ± 1.26 | 77.53 ± 1.03 |
| FlexCon(s)      | 93.84 ± 1.61                 | 95.56 ± 1.42 | 95.55 ± 1.42 | 95.38 ± 1.25 | 95.58 ± 1.32 |
| FlexCon-C1(s)   | 68.37 ± 1.91                 | 73.22 ± 1.46 | 73.90 ± 1.17 | 73.17 ± 1.58 | 78.38 ± 1.25 |
| FlexCon-C1(v)   | 88.59 ± 1.67                 | 82.55 ± 1.49 | 81.35 ± 1.32 | 77.77 ± 1.30 | 78.67 ± 1.37 |
| FlexCon-C2      | 88.85 ± 1.63                 | 74.07 ± 1.35 | 75.88 ± 1.19 | 77.54 ± 1.29 | 78.68 ± 1.29 |
| G               | 97.59 ± 1.48                 | 85.94 ± 1.38 | 78.98 ± 1.38 | 80.96 ± 1.39 | 81.97 ± 1.03 |

It is important to emphasize that, while using Naive Bayes, all the proposed methods (FlexCon-G, FlexCon and FlexCon-C) obtained better accuracy than the Original ST and the ST Fixed Threshold, when the percentage of instances initially labeled was 5%. In this way, it is possible that these methods adapt well in real-world datasets which have a small number of labeled instances. Furthermore, unlike the Naive Bayes classifier, when using a decision tree, the methods performed better than the Original ST and the ST Fixed Threshold when using 25% of instances labeled at the beginning of the process.

Still based on the data presented in Table 2, when analyzing the areas shaded in yellow, it is possible to verify that using Naive Bayes, decision tree, Ripper and k-NN the proposed methods are better than the ST Fixed Threshold at 70% (21 of 30), 76.66% (23 of 30), 83.33% (25 of 30) and 43.33% (13 of 30) of cases, respectively. In addition, it is observed that the proposed methods obtained better performance than the ST Fixed Threshold using the Naive Bayes classifiers, decision tree and Ripper with 5%, 10%, 20% and 25% of the data initially labeled. These results demonstrate that the methods, when using these classifiers, adapt well to both the highest and lowest values of percentages of instances initially labeled.

Table 3 identifies the method which achieved the best performance in relation to the others for each percentage of initially labeled.

**TABLE 3.** Methods with a better performance in comparison to the others for each percentage of initially instances labeled

| Classifier | % instances initially labeled | 5%          | 10%          | 15%          | 20%          | 25%          |
|------------|------------------------------|-------------|-------------|-------------|-------------|-------------|
| Naive Bayes|                             |             |             |             |             |             |
| Self-trained|                            |             |             |             |             |             |
| Decision Tree|                           |             |             |             |             |             |
| Ripper     |                             |             |             |             |             |             |
| k-NN       |                             |             |             |             |             |             |
| G          |                             |             |             |             |             |             |
| FlexCon(s) |                             |             |             |             |             |             |
| FlexCon-C1(s) |                          |             |             |             |             |             |
| FlexCon-C1(v) |                          |             |             |             |             |             |
| FlexCon-C2 |                             |             |             |             |             |             |
| FlexCon-C1(v) |                          |             |             |             |             |             |
| FlexCon-C2 |                             |             |             |             |             |             |
| G          |                             |             |             |             |             |             |
| FlexCon(s) |                             |             |             |             |             |             |
| FlexCon-C1(s) |                          |             |             |             |             |             |
| FlexCon-C1(v) |                          |             |             |             |             |             |
| FlexCon-C2 |                             |             |             |             |             |             |
| FlexCon-C1(v) |                          |             |             |             |             |             |
| FlexCon-C2 |                             |             |             |             |             |             |
| G          |                             |             |             |             |             |             |

Given the above considerations, it is clear that for the Naive Bayes and Ripper classifiers, the best method is always one of those proposed in this article. While for the decision tree and k-NN the proposed methods stood out as better in 4 of the 5 percentages of instances initially labeled.

Table 2 presents the standard deviation, which was calculated considering the average results for each dataset used in the experiments; after this, we calculated the average accuracy and standard deviation over all datasets. Thus, the high values of standard deviation can be justified because the datasets have different characteristics and, consequently, divergent accuracy. In addition, it is noticed that the lowest values of the standard deviation are identified when using the highest percentages of instances initially labeled, otherwise the highest standard deviations are located in the lowest percentages, except for the Naive Bayes algorithm, the values of each are similar for all percentages of instances initially labeled.
Figure 7 presents an image which indicates the number of times that each method achieved the best performance in relation to all other methods, according to each percentage of instances initially labeled and each classifier. To explain it better, consider the following example, the FlexCon-C1(s) obtained greater accuracy than the other methods in two cases: 1) with the Naive Bayes classifier using 5% of the data initially labeled; 2) using $k$-NN as classifier, and 10% of the data initially labeled; so this method appears on the graph with the bar at number two on the $y$ axis. Given the above, it is possible to observe that FlexCon-C2 stands out with superior accuracy in 5 out of 20 cases, 4 with DT and Ripper (using 10% and 25%) and 1 with NB (using 15%). Then, both methods FlexCon(s) and FlexCon(v) perform best in 4 of the 20 cases, and FlexCon-C1(v) in 3 cases.

After evaluating the performance of the methods separated by classifier, the results of each method are analyzed separately. As a way to explore all the results of each of the methods described in this work, Figures 8 and 9 present boxplot-type graphs, produced from the 300 accuracy measures (10 repetitions of each of the 30 datasets) achieved by each method, using self-training. These graphs were organized by method, so each graph has the accuracy of a single method, separated by classifiers and percentages of instances labeled at the beginning of the process. In these graphs, the $x$ axis contains the percentage of instances labeled at the beginning of the process (5%, 10%, 15%, 20%, 25%), while the accuracy values are on the $y$ axis. In addition, as each graph has the performance of the four classifiers, they were separated by colors: red, blue, yellow and green, which represent, respectively, Naive Bayes, decision tree, Ripper and $k$-NN.

By analyzing these graphs, considering the percentages of instances initially labeled, it seems clear that, in all methods, the accuracy values rise as the percentage of labeled instances increases. In other words, the higher the percentage of instances initially labeled, the better the performance of each method for all classifiers. Given the above, it is possible to observe that the greatest accuracy is obtained using 25% of the data initially labeled, for all methods (100% of cases).

Additionally, it is noticeable that the classifiers with the greatest difference in accuracy from the smallest to the highest percentage of instances initially labeled are decision tree and Ripper, in most cases. Naive Bayes and $k$-NN have similar performance for all percentages of instances labeled at the beginning of the process. In addition, it becomes evident that Naive Bayes was the classifier with the highest amount of outliers in all methods.

Table 4 shows a ranking with the datasets that achieved the best and the worst performance, using the average accuracy, during the experiments using the original self-training and the proposed algorithms. For simplification purposes, we present only three datasets with the best and worst ranking performance. The data was split horizontally for classifiers and vertically for each approach analyzed in this paper (ST-Original, STFT, and FlexCon-Family). In this case, the results of six variations proposed method were aggregated (using majority vote), grouped by an only column, and named FlexCon-Family.

In general, although the FlexCon-Family algorithms have been the best performance than the traditional approaches, we observed a similar behavior tendency with the ranking of datasets, independently to the evaluated method. In other words, the datasets that achieved the best results using the FlexCon-Family algorithms are practically the same that achieved the best results using the traditional approaches, with minor variations.

On the other hand, we observed that the performance of each dataset in the ranking presented is the most relationship...
with the classifier used than the approach used. The alteration of the classifier has a direct influence on the accuracy of the process. With the Ripper and k-NN classifiers, according to Table 4, we observed that the best and the worst datasets are the same, independently of the approach used.

2) Performance Analysis with Co-training
The results related to the performance of each method using the co-training algorithm with the classifiers Naive Bayes, decision tree, Ripper and k-NN are presented in Table 5. According to the data presented, it is noticeable that all methods obtained greater accuracy than the Original CT in 100% of the cases (values in bold).

Comparing the accuracy of the proposed methods with those of CT Fixed Threshold (cells shaded in yellow in Table 5), Naive Bayes achieved better results in 3 out of 30 cases, and while using the other classifiers, none of the proposed methods exceeded the accuracy of this method. This result can be justified by the number of instances labeled during the process, as the proposed methods classify all of the unlabeled instances, while the CT Fixed Threshold does not classify instances whose confidence rate is lower than the threshold initially defined.

This behavior of the proposed methods implies the possibility of including instances with low confidence, which can negatively influence the prediction of the classifiers. Otherwise, CT Fixed Threshold procedure may result in the inclusion of too few instances in the training set. In this
TABLE 4. The best and worst datasets with self-training

| Classifiers | Ranking | ST-Original | STFT | FlexCon |
|-------------|---------|-------------|------|---------|
| Naive Bayes |         |             |      |         |
|             | 1       | 1           | 1    | 1       |
|             | 2       | 2           | 2    | 2       |
|             | 3       | 3           | 3    | 3       |
|             | 28      | 28          | 28   | 28      |
|             | 29      | 29          | 29   | 29      |
|             | 30      | 30          | 30   | 30      |
| Decision Tree |         |             |      |         |
|             | 1       | 1           | 1    | 1       |
|             | 2       | 2           | 2    | 2       |
|             | 3       | 3           | 3    | 3       |
|             | 28      | 28          | 28   | 28      |
|             | 29      | 29          | 29   | 29      |
|             | 30      | 30          | 30   | 30      |
| Ripper      |         |             |      |         |
|             | 1       | 1           | 1    | 1       |
|             | 2       | 2           | 2    | 2       |
|             | 3       | 3           | 3    | 3       |
|             | 28      | 28          | 28   | 28      |
|             | 29      | 29          | 29   | 29      |
|             | 30      | 30          | 30   | 30      |
| k-NN        |         |             |      |         |
|             | 1       | 1           | 1    | 1       |
|             | 2       | 2           | 2    | 2       |
|             | 3       | 3           | 3    | 3       |
|             | 28      | 28          | 28   | 28      |
|             | 29      | 29          | 29   | 29      |
|             | 30      | 30          | 30   | 30      |

TABLE 5. Average Accuracy of all methods using Naive Bayes, decision tree, Ripper and k-NN classifiers with co-training

| Method                  | % of instances initially labeled | 5% | 10% | 15% | 20% | 25% |
|-------------------------|----------------------------------|----|-----|-----|-----|-----|
| ST-Original             |                                  |    |     |     |     |     |
| Naive Bayes             |                                  |    |     |     |     |     |
| STFT                    |                                  |    |     |     |     |     |
| FlexCon                 |                                  |    |     |     |     |     |
| Decision Tree           |                                  |    |     |     |     |     |
| Ripper                  |                                  |    |     |     |     |     |
| k-NN                    |                                  |    |     |     |     |     |

FIGURE 10. Average Percentage of instances labeled using the static threshold methods and the methods proposed.

By analyzing Figure 10, it is possible to verify that ST Fixed Threshold - STFT (left graph) labels approximately 80% of the instances using the fixed threshold at 95%. Using this same threshold value, CT Fixed Threshold - CTFT (graph on the right) labels the worst and best cases, respectively, 30% and 70% of the instances of the unlabeled dataset. On the other hand, the proposed methods, which use a flexible threshold, label the entire set of unlabeled data and begin their labeling process with a threshold value of 95%; however, they need to decrease this value to include the remaining instances. Given the above, it is possible to reaffirm that the small number of instances labeled by CT Fixed Threshold may have positively influenced the prediction of the classifiers due to the construction of a training set formed only by instances whose prediction is reliable.

Additionally, a sample experiment was carried out using the FlexCon(s) method (Stop Criterion - SC) in which the labeling process is interrupted at the moment when there are no new instances to be labeled, instead of labeling all instances of the unlabeled dataset. The accuracy of that experiment is shown in Table 6 in the row called FlexCon(s)-SC. The other rows of the referred table were replicated from Table 5 for comparison of results. The values in bold way, the training set can contain only those instances whose prediction is reliable, positively affecting the prediction of the classifiers.

To prove the justification above, two graphs were created, shown in Figure 10, which present the average percentage of instances labeled using the methods with fixed threshold and the ones proposed. In these graphs, the x axis represents the percentage of instances initially labeled and the y axis represents the average percentage of instances included in the labeled dataset during the labeling process. The red bars indicate 100% of instances labeled by the proposed methods, while the colors green, blue, yellow and brown represent the fixed threshold method using the classifiers Naive Bayes, decision tree, Ripper and k-NN, respectively.
represents the cases in which FlexCon(s)-SC achieved greater accuracy than the Original CT and FlexCon(s), while the cells shaded in yellow reflect the situations in which FlexCon(s)-SC performed higher than CT Fixed Threshold.

### TABLE 6. Average Accuracy of FlexCon(s) using Stop Criterion (co-training)

| Method          | 5%     | 10%    | 15%    | 20%    | 25%    |
|-----------------|--------|--------|--------|--------|--------|
| **Naive Bayes** |        |        |        |        |        |
| CT Original     | 63.90  | 64.73  | 65.32  | 65.63  | 65.89  |
| CT Fixed        | 63.90  | 65.63  | 65.89  | 65.98  | 66.64  |
| FlexCon(s)-SC   | 64.65  | 66.64  | 67.15  | 67.64  | 68.34  |

**Average Accuracy of FlexCon(s) using Stop Criterion (co-training)**

Analyzing the data in Table 6, it can be seen that the FlexCon(s)-SC method achieved greater accuracy than the CT Original and FlexCon(s) methods in all cases (values marked in bold). In addition, FlexCon(s)-SC performed better than the CT Fixed Threshold in 3 and 5 of the 5 percentages of instances initially labeled using Naive Bayes and Ripper, respectively (yellow shaded cells). In all other cases, Flexcon(s)-SC showed a performance similar to CT Fixed Threshold’s. Considering the above, it is possible to propose that the methods may be labeling instances whose confidence in the prediction is very low and therefore their performance is not being superior to CT Fixed Threshold’s. However, there is an evident trade-off in which the proposed methods label all instances of the unlabeled dataset, although their performance in relation to CT Fixed Threshold decreases.

Analyzing the results of Table 5, comparing only the accuracy of the proposed methods without considering CT Original and CT Fixed Threshold, it is possible to observe that FlexCon-C2, using Naive Bayes, FlexCon(v), using decision tree and Ripper and FlexCon-C1(v) with k-NN, were the ones that showed the best performance. In other words, these methods were more accurate than the others in most of the 5 percentages of instances initially labeled.

Following the same dynamics as self-training, the standard deviation shown in Table 5 was calculated considering the average accuracy of the 30 datasets used in the experiments of this work, therefore, the high values of this measure can be justified due to the datasets having different characteristics and consequently different performances. In addition, it is noticeable that, while using the Naive Bayes and Ripper algorithms, the standard deviation is similar for all percentages of instances initially labeled, whereas for Decision Tree and k-NN the values are equivalent when the percentages are greater than 5%.

Figure 11 presents a graph which indicates how many times each method achieved better performance, according to each percentage of instances initially labeled and each classifier. Analyzing the data in general, it is evident that FlexCon(v) stands out with the best accuracy in 9 out of 20 cases, 3 with NB, DT and Ripper (using5%), 2 with DT and k-NN (using 10%), 2 with DT and 2 with Ripper (both using 15% and 20%). Then, the FlexCon-C2 method with the best performance in 5 out of 20 cases, FlexCon-C1(v) in 4 cases, FlexCon(s) with 2 cases and FlexCon-C1(s) with 1 case.

Following the same approach as with self-training, for co-training, boxplot-type graphics were also generated, produced from the 300 accuracy (10 repetitions of each of the 30 datasets) achieved by each method. Such graphs were organized by method and are shown in Figures 12 and 13. Given the above, each graph has the accuracy of a single method, separated by classifiers and percentages of instances labeled at the beginning of the process. In these graphs, the x axis contains the percentage of instances initially labeled (5%, 10%, 15%, 20%, 25%), and the y axis are the accuracy values. In addition, each graph has the performance of the four classifiers, which were separated by colors: red, blue, yellow and green represent, respectively, Naive Bayes, decision tree, Ripper and k-NN.

By analyzing the graphs presented above, considering the percentages of instances initially labeled, it becomes evident that the results are similar to the ones with self-training, for the higher the percentage, the higher the accuracy values. Given the above, it is possible to notice that the greatest accuracy is obtained using 25% of the data initially labeled, for all methods (100% of cases).

Additionally, it is noticeable that the classifiers with the greatest difference in accuracy from the smallest to the highest percentage of instances initially labeled are decision tree and Ripper, in most cases. Naive Bayes and k-NN have similar performance for all percentages of instances labeled at the beginning of the process. Now considering the divergent values, it becomes apparent that Naive Bayes has a large amount of them in all methods and percentages of initially labeled data, except in the CT Original and FlexCon-G. k-
NN, on the other hand, obtained discrepant accuracy with several methods, namely: FlexCon-G, FlexCon(s and v) and FlexCon-C1(s and v).

Similarly, as presented in the previous section, Table 7 shows the rank of the datasets using the average accuracy during the experiments using original co-training and FlexCon methods. Table 7 shows a ranking with the datasets that achieved the best and the worst performance, using the average accuracy, during the experiments using the original co-training and the proposed algorithms. For simplification purposes, we present only three datasets with the best and worst ranking performance. The data was split horizontally for classifiers and vertically for each approach analyzed in this paper (ST-Original, STFT, and FlexCon-Family). In this case, the results of six variations proposed method were aggregated (using majority vote), grouped by an only column, and named FlexCon-Family.

Like the self-training, in general, although the FlexCon-Family algorithms have been the best performance than the traditional approaches, we observed a similar tendency toward raking datasets independently to the evaluated method. In other words, the datasets that achieved the best results using the FlexCon-Family algorithms are practically the same that achieved the best results using the traditional approaches, with minor variations.

From these results, analyzing the ranking of each dataset concerning the classifiers, it is possible to observe that the best and the worst datasets are the same for all classifiers, except for some variations using k-NN and Ripper. Using Naive Bayes and decision tree, we observed in Table 7 that both the first three and last three datasets of ranking are the same, independently that the approach used, except for any changes in ranking positioning.

**TABLE 7.** The best and worst datasets with co-training

| Classifiers | Ranking | CT-Original | CTFT | FlexCon-Family |
|-------------|---------|-------------|------|---------------|
| Naive Bayes |         |             |      |               |
|             | 1       | Will        | Will | Will          |
|             | 2       | Tenseon     | Tenseon | Tenseon      |
|             | 3       | Mushroom    | Mushroom | Mushroom   |
|             | 4       |                 |       |               |
| Decision Tree |       |             |      |               |
|             | 1       | Mushroom    | Mushroom | Mushroom   |
|             | 2       | Viele      | Will | Will          |
|             | 3       |                 |       |               |
| Ripper      |         |             |      |               |
|             | 1       | Ozone        | Ozone | Ozone        |
|             | 2       |                 |       |               |
| k-NN        |         |             |      |               |
|             | 1       | Ozone        | Ozone | Ozone        |
|             | 2       |                 |       |               |

B. STATISTICAL ANALYSIS

After evaluating the performance of each method using Naive Bayes, decision tree, Ripper and k-NN as classification algorithms in the labeling procedure, a statistical analysis of the results was performed. As explained earlier, Friedmann and post-hoc Nemenyi tests were used to compare the performance of different methods applied to different datasets.

The statistical test was applied separately for the self-training and co-training algorithms. However, each percentage of instances initially labeled and the four classifiers were considered together in order to facilitate the visualization of the results. First, Friedmann test was performed, which showed that the performances of the different methods are statistically different. The significant difference was detected by Friedmann test, with p-value < 0.001, for all proportions of data initially labeled.

Given the statistical difference presented by Friedmann test, the paired test post-hoc Nemenyi was then, applied to compare the different methods, two by two, in each percentage of instances initially labeled. The result of this test is detailed in the next sections using the Critical difference diagrams presented in Figures 14 and 15. The methods located on the left are considered better than those on the right, from a statistical point of view. The methods connected by the horizontal bar are those with a similar performance and therefore have no statistical difference. Otherwise, the methods that are not linked by the horizontal bar are statistically different, with the method on the left being superior to the one on the right.
1) Statistical Analysis with Self-training

Figure 14 presents the critical difference diagrams obtained from the post-hoc Nemenyi statistical test for the self-training algorithm. This figure contains the diagrams separated by the percentage of instances initially labeled. The first observation that can be made regarding these diagrams is that methods FlexCon and FlexCon-C obtained the best rankings, i.e., they appear more to the left in most cases. FlexCon-C2 stands out as superior from a statistical point of view, as it is always to the left of the diagram and has a critical difference in relation to at least one of the methods in 4 of the 5 percentages of instances initially labeled. In addition, this method achieved the lowest ranking when it used the lowest percentages of instances initially labeled (5%, 10% and 15%).

Analyzing the diagrams in Figure 14, it is clear that the difference between all methods and the original are statistically significant in all percentages of instances initially labeled, except for FlexCon-G with 5%, 15% and 25%. Considering the statistical performance of the proposed methods in relation to the fixed threshold method, it is noticeable that the proposed methods are always positioned more to the left. This means that the proposed methods have better ranking than the fixed threshold, although they are statistically similar.
FIGURE 14. Statistical results of the methods by percentage of instances initially labeled (self-training)
FIGURE 15. Statistical results of the methods by percentage of instances initially labeled (co-training)
2) Statistical Analysis with Co-training

Figure 15 presents the critical difference diagrams obtained from the post-hoc Nemenyi statistical test for the co-training algorithm. As with self-training, diagrams are separated by the percentage of instances initially labeled. Analyzing the diagrams of that figure, it is clear that the difference between all methods and the original are statistically significant, except FlexCon-G with 5% and 20%.

By analyzing the diagram that uses 5% of instances initially labeled, it is apparent that method FlexCon(s and v) is statistically similar to the fixed threshold method. This can be considered a good result, because in addition to this similarity, FlexCon has adapted well, from a statistical point of view, using few instances initially labeled. As justified previously, the method that uses fixed threshold achieved better performance than the others, due to the fact that it labeled fewer instances. Therefore, its training set can be formed only by instances with high reliability. The same justifies its good position in the ranking of the statistical test.

Corroborating the performance analysis performed previously, despite the fixed threshold obtaining better statistical performance, it is possible to notice that methods FlexCon and FlexCon-C, in all diagrams, obtained good positions in the ranking, i.e., they were positioned on the left of the diagram.

The results (accuracy and standard deviation) from the application of self-training showed that, with the Naive Bayes and k-NN classifiers, all methods achieved better performance than ST Original. In addition, most methods, 68.33% (82 of 120), obtained greater accuracy than ST Fixed Threshold. On the other hand, using co-training, all four classifiers performed better than ST Original.

The evaluation from the statistical point of view was performed using Friedmann and post-hoc Nemenyi tests, which compared the methods proposed to the original ones, separating them by percentage of instances initially labeled. From this investigation, it was observed that the proposed methods are better from a statistical point of view in most cases. In summary, the results presented in this study suggest that the proposed methods have, on average, superior performance to the original methods, both in terms of accuracy and standard deviation, and in statistical analysis.

VIII. CONCLUSION

The present work is included in the machine learning field, more specifically, semi-supervised learning. One of the main limitations of semi-supervised learning algorithms is related to the selection of new instances to be included in the labeled dataset. In this context, several studies have been carried out to try to solve this problem. However, none of them employed the approach presented in this work, which uses a dynamic confidence threshold, to include new instances in the training set at each iteration.

This work proposed three methods, namely: FlexCon-G, FlexCon and FlexCon-C, which include a dynamic confidence rate calculation for choosing labels used in the semi-supervised labeling process with self-training and co-training. In this way, while the original self-training and co-training (as well as their derivations) use a static procedure to include new instances in the labeled dataset, methods FlexCon-G, FlexCon and FlexCon-C aim to make the labeling procedure more flexible. Such a strategy allows the proposed methods to be able to more deeply explore the full potential of a semi-supervised technique.

To assess the feasibility of this proposal, experiments were carried out using 30 sets of classification data, organized in 5 different scenarios, with regard to the proportion of instances initially labeled (5%, 10%, 15%, 20% and 25%). In addition, four different classification algorithms were used in the self-training procedure, Naive Bayes, decision tree, Ripper and k-NN.

The results of the experiments were evaluated from two perspectives:

1) performance of the methods regarding accuracy and standard deviation;
2) statistical analysis.

Based on this analysis, it was possible to conclude that the proposed methods – the objective of which is to make the confidence rate dynamic – were very promising, as they, in most cases, improved the classification accuracy when compared to the methods in their original form and in the fixed threshold form.

Finally, by exploring the accuracy of the methods proposed by the classifier, it was possible to conclude that, when using the classifiers Naive Bayes and k-NN, the proposed methods stand out from the others, as they obtained greater accuracy than the original methods, both for self-training and for co-training. By investigating the performance of the methods according to the percentage of instances initially labeled, it is apparent that the greatest accuracy is achieved when using 25% in all cases, except with self-training using the Naive Bayes classifier.

The following are some works that can be developed in future research involving the proposed methods:

- The methods were applied to the self-training and co-training algorithms, however there is the possibility of using other semi-supervised learning algorithms, as well as other classifiers;
- Investigation of new strategies to address or avoid the inclusion unreliable instances, i.e., those whose confidence rate in the prediction is very low;
- Development of a data stratification process to include new instances in the labeled dataset, which must be applied using co-training;
- In this work, five percentages of instances initially labeled were used, therefore, other percentages can be used and compared with this research;
- It is possible to conduct an investigation of the best value for the initial confidence threshold;
- The confidence parameter can be replaced by confidence intervals instead of using a single threshold value, allowing for greater flexibility in the algorithm.
To compare the performance of our methods with other approaches, for example, oversampling techniques.

REFERENCES

[1] J. Gama, K. Ferilli, A. Lorena, and A. De Carvalho, Artificial Intelligence: a machine learning approach. In Portuguese: Inteligência artificial: uma abordagem de aprendizado de máquina. Grupo Gen - LTC, 2011. [Online]. Available: https://books.google.com.br/books?id=4DwleAEACAAJ

[2] F. Herrera, F. Charte, A. J. Rivera, and M. J. del Jesus, Multiclass Classification: Problem Analysis, Metrics and Techniques. Springer, 2016.

[3] O. Chapelle, B. Scholkopf, and A. Zien, Semi-Supervised Learning, O. Chapelle, B. Scholkopf, and A. Zien, Eds. The MIT Press, 2006.

[4] X. Zhu and A. B. Goldberg, “Introduction to semi-supervised learning,” Synthesis Lectures on Artificial Intelligence and Machine Learning, vol. 3, no. 1, pp. 1–130, 2009.

[5] H. Gan, N. Sang, R. Huang, X. Tong, and Z. Dan, “Using clustering analysis to improve semi-supervised classification,” Neurocomputing, vol. 101, pp. 290 – 298, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231210006601

[6] Y. Tao, D. Zhang, S. Cheng, and X. Tang, “Improving semi-supervised self-training with embedded manifold transcription,” Transactions of the Institute of Measurement and Control, vol. 0, no. 0, p. 0142331216658091, 2016. [Online]. Available: http://dx.doi.org/10.1177/0142331216658091

[7] D. Wu, M. sheng Shang, X. Luo, J. Xu, H. yong Yan, W. hui Deng, and G. yin Wang, “Self-training semi-supervised classification based on density peaks of data,” Neurocomputing, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0925231217309608

[8] K. M. O. Vale, A. M. P. Canuto, A. M. Santos, F. L. Gorgonio, A. M. Tavares, A. C. Gorgonio, and C. T. Alves, “Automatic adjustment of confidence values in self-training semi-supervised method,” in Proceedings of the International Joint Conference on Neural Networks (IJCNN 2018), 2018.

[9] D. Kim, D. Sea, S. Cho, and P. Kang, “Multi-to-code for document classification using various document representations: TF-idf, lda and doc2vec,” Information Sciences, 2019.

[10] A. C. Gorgonio, A. M. de P. Canuto, K. M. O. Vale, and F. L. Gorgonio, “A semi-supervised based framework for data stream classification in non-stationary environments,” in 2020 International Joint Conference on Neural Networks (IJCNN), 2020, pp. 1–8.

[11] K. M. O. Vale, F. L. Gorgonio, Y. N. Araújo, A. C. Gorgonio, and A. M. de P. Canuto, “A co-training-based algorithm using confidence values to select instances,” in 2020 International Joint Conference on Neural Networks (IJCNN), 2020, pp. 1–7.

[12] D. Yarowsky, “Unsupervised word sense disambiguation rivaling supervised methods,” in Proceedings of the 33rd Annual Meeting on Association for Computational Linguistics, ser. ACL ’95. Stroudsburg, PA, USA: Association for Computational Linguistics, 1995, pp. 189–196. [Online]. Available: https://doi.org/10.3115/981658.981684

[13] A. Blum and T. Mitchell, “Combining labeled and unlabeled data with co-training,” in Proceedings of the Eleventh Annual Conference on Computational Learning Theory, ser. COLT ’98. New York, NY, USA: ACM, 1998, pp. 92–100. [Online]. Available: http://doi.acm.org/10.1145/279943.279962

[14] I. Izonin, R. Tkachenko, N. Shakhovska, and N. Lotoshynska, “The additive input-doubleing method based on the SVR with nonlinear kernels: Small data approach,” IEEE Transactions on Knowledge and Data Engineering, vol. 28, no. 7, pp. 1864–1877, July 2016.

[15] J. E. Van Engelen and H. H. Hoos, “A survey on semi-supervised learning,” Machine Learning, vol. 109, no. 2, pp. 373–440, 2020.

[16] A. Albalate and W. Minker, Semi-Supervised and Unsupervised Machine Learning - Novel Experiments with Weka, 2011.

[17] M. C. Monard and J. A. Baranauskas, Sistemas Inteligentes: Fundamentos e Aplicações. Manole, 2003, ch. Conceitos sobre Aprendizado de Máquina, p. 89 114.

[18] E. Alpaydin, Introduction to Machine Learning. The MIT Press, 2010.

[19] N. P. Pathy, Artificial intelligence and intelligent systems. New Delhi, 2005.

[20] T. M. Mitchell, Machine Learning. McGraw-Hill, 1997.

[21] L. Breiman, Friedman, R. A. Olshen, and C. J. Stone, Classification and Regression Trees. New York: CHAPMAN & HALL/CRC, 1984.

[22] W. W. Cohen, “Fast effective rule induction,” in Proceedings of the Twelfth International Conference on International Conference on Machine Learning, ser. ICML’95. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1995, pp. 115–123. [Online]. Available: http://dl.acm.org/citation.cfm?id=3091622.3091637

[23] L. J. Kuncheva, Combining Pattern Classifiers: Methods and Algorithms, 2nd ed. Wiley Publishing, 2014.

[24] D. S. Nascimento, A. L. V. Coelho, and A. M. P. Canuto, “Integrating complementary techniques for promoting diversity in classifier ensembles: A systematic study,” in Neurocomputing (Amsterdam), 2014.

[25] O. Gharroudi, “Ensemble multi-label learning in supervised and semi-supervised settings,” Theses, Université de Lyon, Dec. 2017. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01765496

[26] F. Herrera, F. Charte, A. J. Rivera, and M. J. del Jesus, Artificial intelligence and intelligent systems. New York: CHAPMAN & HALL/CRC, 1984.

[27] L. I. Kuncheva, Combining Pattern Classifiers: Methods and Algorithms, 2nd ed. Wiley Publishing, 2014.

[28] Z. Wang, B. Du, and Y. Guo, “Domain adaptation with neural embedding matching,” IEEE Transactions on Neural Networks and Learning Systems, vol. 31, no. 7, pp. 2387–2397, 2020.

[29] Z. Wang, B. Du, W. Tu, L. Zhang, and D. Tao, “Incorporating distribution matching into uncertainty for multiple kernel active learning,” IEEE Transactions on Knowledge and Data Engineering, vol. 33, no. 1, pp. 128–142, 2021.

[30] Z. Wang, B. Du, L. Zhang, L. Zhang, and X. Jia, “A novel semisupervised active-learning algorithm for hyperspectral image classification,” IEEE Transactions on Geoscience and Remote Sensing, vol. 55, no. 6, pp. 3071–3083, 2017.

[31] J. Tanha, M. van Someren, and H. Afsarmanesh, “Semi-supervised self-training for decision tree classifiers,” International Journal of Machine Learning and Cybernetics, vol. 8, no. 1, pp. 355–370, Feb 2017. [Online]. Available: https://doi.org/10.1007/s13042-015-0328-7

[32] S. Samiapan and R. J. Moorhead, “Semi-supervised co-training and active learning framework for hyperspectral image classification,” in 2015 IEEE International Geoscience and Remote Sensing Symposium (IGARSS), July 2015, pp. 401–404.

[33] B. Arroyoñada, T. B. Adji, and I. Hidayah, “Semi-supervised learning approach for indonesian named entity recognition (ner) using co-training algorithm,” in 2016 International Seminar on Intelligent Technology and Its Applications (ISITA), July 2016, pp. 7–12.

[34] N. D. Hoang, V. T. N. Chau, and N. H. Phuong, “Combining transfer learning and co-training for student classification in an academic credit card application system,” in 2016 IEEE RIVF International Conference on Computing Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), Nov 2016, pp. 55–60.

[35] J. Chen, F. Liu, B. Avci, X. Wu, Y. Liang, and S. Jha, “Detecting errors and estimating accuracy on unlabeled data with self-training ensembles,” 2021.

[36] C. Wang, Z. Xu, S. Wang, and H. Zhang, “Semi-supervised classification framework of hyperspectral images based on the fusion evidence entropy,” Multimedia Tools and Applications, vol. 76, no. 18, pp. 10041–10044, Apr 2017. [Online]. Available: https://doi.org/10.1007/s11042-017-4866-x

[37] S. Qiao, W. Shen, Z. Zhang, B. Wang, and A. Yuille, “Deep co-training for semi-supervised image recognition,” in The European Conference on Computer Vision (ECCV), September 2018.

[38] Y. Chen, T. Pan, and S. Chen, “Development of co-training support vector machine model for semi-supervised classification,” in 2017 5th Chinese Control Conference (CCC), July 2017, pp. 401–404.

[39] V. Frinken, A. Fischer, M. Baumgartner, and H. Bunke, “Keyword spotting for self-training of bi stm nn based handwriting recognition systems,” Pattern Recognition, vol. 47, no. 3, pp. 1073 – 1082, 2014.
handwriting Recognition and other PR Applications. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320313002823

[43] B. Wang, B. Spencer, C. X. Ling, and H. Zhang, "Semi-supervised Self-training for Sentence Subjectivity Classification," Springer Berlin Heidelberg, 2008, pp. 344–355.

[44] S. Aghababaei and M. Makrehchi, “Interpolative self-training approach for sentiment analysis,” in 2016 International Conference on Behavioral, Economic and Socio-cultural Computing (BESC), Nov 2016, pp. 1–6.

[45] A. Salazar, G. Safont, L. Vergara, and E. Vidal, “Pattern recognition techniques for provenance classification of archaeological ceramics using ultrasonics,” Pattern Recognit. Lett., vol. 135, pp. 441–450, 2020. [Online]. Available: https://doi.org/10.1016/j.patrec.2020.04.013

[46] D. H. Wolpert, “Stacked generalization,” Neural Networks, vol. 5, no. 2, pp. 241 – 259, 1992.

[47] D. Dheeru and E. Karra Taniskidou, “UCI machine learning repository,” 2017. [Online]. Available: http://archive.ics.uci.edu/ml

[48] J. Alcala-Fdez, A. Fernandez, J. Luengo, J. Derrac, and S. Garcia, “Keel data-mining software tool: Data set repository, integration of algorithms and experimental analysis framework.” Multiple-Valued Logic and Soft Computing, vol. 17, no. 2-3, pp. 255–287, 2011. [Online]. Available: https://dblp.uni-trier.de/db/journals/mvl/mvl17.html

[49] J. W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes, “Using the ADAP learning algorithm to forecast the onset of diabetes mellitus,” in Proceedings of the Symposium on Computer Applications and Medical Care. IEEE Computer Society Press, 1988, pp. 261–265.

[50] L. Breiman, “Bias, variance, and arcing classifiers,” Statistics Department, University of California at Berkeley, Tech. Rep. 460, 1996.

[51] I. H. Witten, E. Frank, and M. A. Hall, Data Mining: Practical Machine Learning Tools and Techniques, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[52] L. Torgo, Data Mining with R: Learning with Case Studies, Second Edition, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017.

[53] E. Theodorsson-Norheim, “Friedman and quade tests: Basic computer program to perform nonparametric two-way analysis of variance and multiple comparisons on ranks of several related samples,” Computers in biology and medicine, vol. 17(2), pp. 85–99, 1987.

KARLIANE M. OVIDIO VALE received her Ph.D. degree from the Federal University of Rio Grande do Norte (UFRN), Brazil, in 2019. Currently, she is an associate professor in the Department of Computing and Technology, UFRN, where develop researchs join at Laboratory of Computational Intelligence Applied to Business (labican.org). Her research interests include the field of computational intelligence, more specifically on machine learning. She has published several articles in scientific journals and conferences.

ARThUR C GORGÔNIO received the B.S. degree in Information Systems from the Federal University of Rio Grande do Norte, Caicó, Brazil, in 2018. He obtained the M.Sc. degree in Systems and Computation at Federal University of Rio Grande do Norte, Natal, Brazil, in 2021. He is currently pursuing the Ph.D degree at the same university. His research interest includes machine learning systems, semi-supervised learning and data-stream classification.

FLAVIUS L. GORGÔNIO was born in Campina Grande, PB, Brazil, in 1970. He received his B.S. degree in Computer Science (1995) and his M.S. degree in Informatics (1999) from the Federal University of Paraíba, Campina Grande, PB. He obtained his doctorate in Electrical and Computer Engineering (2009) from the Federal University of Rio Grande do Norte, Natal, RN. Currently, he is an associate professor at the Federal University of Rio Grande do Norte, Caicó, RN, Brazil, linked to the Department of Computing and Technology, where he coordinates the Laboratory of Computational Intelligence Applied to Business (labican.org). His research interests include computer science, programming languages, computational intelligence and internet of things.

ANNE M P CANUTO received her Ph.D. degree from the University of Kent, in 2001. Currently, she is an associate professor in the Informatics and applied mathematics department, Federal University of Rio Grande do Norte, Brazil. She has published several articles in scientific journals and conferences. Her interests include pattern recognition, clustering algorithms, biometrics, classifier combination, semi-supervised learning and multi-agent systems.

***