Information Fusion 89 (2023) 567–587

Contents lists available at ScienceDirect

Information Fusion

journal homepage: www.elsevier.com/locate/inffus

Full length article

Exploring diversity in data complexity and classifier decision spaces for pool generation

Marcos Monteiro Jr [a,*], Alceu S. Britto Jr [a,b], Jean P. Barddal [a], Luiz S. Oliveira [c], Robert Sabourin [d]

[a] Pontifícia Universidade Católica do Paraná (PUCPR), Curitiba, PR, Brazil
[b] State University of Ponta Grossa (UEPG), Ponta Grossa, PR, Brazil
[c] Federal University of Paraná (UFPR), Curitiba, PR, Brazil
[d] École de Technologie Supérieure (ÉTS), Université du Québec, Montréal, QC, Canada

ARTICLE INFO

Keywords:
Classifier pool generation
Diversity
Data complexity measures

ABSTRACT

This paper introduces a novel method for classifier pool generation in which a two-level strategy explores diversity in both data complexity and classifier decision spaces. The rationale is to induce pool members using data subsets representing subproblems with different difficulties while promoting diversity in classifiers' decisions. Two possible variants of the proposed method with a focus on *maximum dispersion* and *maximum accuracy* are presented. These differ in the property used to define the best pool of classifiers provided by an optimization process. A robust experimental protocol encompassing 28 classification datasets shows that the proposed pool generation provided the best accuracy on 327 over 336 experiments (97.3%) when compared to well-known pool generation methods to provide multiple classifier systems with and without dynamic selection.

1. Introduction

Multiple classifier systems (MCS) are often an alternative to avoid the risk of choosing a single model to cover the entire feature space of classification problems showing high complexity in terms of difficulty to separate classes. The creation of an MCS usually encompasses three steps: (i) pool generation, (ii) classifier selection, and (iii) classifier aggregation [1]. The first step, which is the focus of this paper, provides a pool of classifiers. The classifier selection step is optional, and can be performed during the MCS training or testing phases, characterizing a static or dynamic selection, respectively. We can find in the literature a large number of selection alternatives for single classifier or ensemble selection using static or dynamic strategies [2]. The aggregation step combines the classifier's decisions using the entire pool or a selected ensemble.

Even though there is no clear correlation between diversity and accuracy [3], pool generation is usually done by exploring the idea of creating diverse classifiers in the sense that they make different mistakes during predictions, expecting to be complementary. Some well-known methods in the literature explore diversity to create homogeneous ensembles given a base inducer. Such successful strategies have in common the manipulation of the problem data, horizontally, vertically, or both. The former trains classifiers in distinct subsets of instances, e.g., Bagging [4] and Boosting [5], while the latter trains the classifiers on samples represented by just part of the original feature space, e.g., Random Subspace [6]. It is also noteworthy approaches like Random Forest [7], which combines these two strategies of data manipulation, training trees on subsets of data that different feature subspaces can represent.

In this paper, we go further in horizontal data manipulation by considering classification complexity measures to drive data subsets creation during the pool generation step. The rationale is to use a sampling process to create data subsets with different levels of classification difficulty, which we can consider as subproblems of the classification problem at hand. To this end, we propose a novel method for pool generation that organizes subsets of data for training the classifiers of a pool using a two-level strategy to promote diversity: (i) the data complexity space and (ii) diversity in the decision space. The first is related to the complexity or difficulty level of the data subset used to train each pool member. As already mentioned, the idea is to train the pool members in data subsets representing subproblems with different difficulties. For this purpose, the sampling strategy uses data complexity measures [8]. The second part of the strategy acts in the output of the generated classifiers using a well-known diversity measure [9]. The motivation for such a two-level diversity strategy is to create classifiers that can adequately deal with specific subproblems individually and, when combined, with the entire problem.

More specifically, this paper aims to describe a new two-step based method for pool generation. The first step involves selecting the most

* Corresponding author.
E-mail addresses: marcos@ppgia.pucpr.br (M. Monteiro Jr), alceu@ppgia.pucpr.br (A.S. Britto Jr), jean.barddal@ppgia.pucpr.br (J.P. Barddal).

https://doi.org/10.1016/j.inffus.2022.09.001
Received 7 September 2021; Received in revised form 30 August 2022; Accepted 2 September 2022
Available online 6 September 2022

appropriate data complexity measures given a classification problem. We consider those metrics showing high dispersion computed among training subsets generated from the original training data. The rationale is to use such metrics later to drive the creation of training subsets in an optimization process looking for diversity in the data complexity space. In the second step, we train the pool of classifiers using subsets (bags) of data optimized w.r.t. their diversities in the complexity space employing the metrics selected in the first step and decision space using the classifiers' output. Such optimization is performed using a multi-objective genetic algorithm, i.e., NSGA-II [10]

Unlike the work previously published in [11], we describe two possible variants of the proposed method with a focus on *maximum dispersion* or *maximum accuracy*. They differ in the property used to define the best generation to output the pool of classifiers. The former provides the pool related to the generation showing the highest diversity in data complexity and classifier decisions. The latter provides the pool connected to the generation with the highest accuracy. Besides, we consider a more robust experimental protocol using different base classifiers for pool generation and comparing with other classical pool generators like Adaboost and RF, and a competitor also oriented by data complexity measures.

In this scenario, we have two main research questions:

1. RQ1: Is the pool generation method oriented by a two-level diversity that explores diversity in the data complexity space and classifier decision spaces competitive against traditional pool generation methods available in the literature?
2. RQ2: Can the pool of classifiers provided by the proposed method contribute positively to the dynamic classifier/ensemble selection methods?

To answer our research questions and evaluate the proposed method, a robust experimental protocol with 28 classification datasets was used to compare our proposal to other pool generation, dynamic classifier selection (DCS), and dynamic ensemble selection (DES) methods. In general, the experimental results have shown that the proposed pool generation provided the best accuracy in 97.3% of the performed experiments that combine all the classifiers in the pool. Besides, the classification pools created are highly competitive when a specific member or a subset of classifiers is dynamically selected.

The contribution of this work is threefold, as follows: (i) A new method for generation of homogeneous pools of classifiers that explore the diversity in the data complexity and classifier decision spaces, which the code is available at https://anonymous.4open.science/r/_PGCDS-2C65/README.md; (ii) A sampling method based on data complexity measures that allow the training of classifiers on subsets of data showing different levels of difficulty; (iii) A positive impact on the performance of methods based on the dynamic selection of classifiers and ensembles.

This paper is organized as follows. Section 2 presents related works considering pool generation and dynamic selection of classifier and ensembles. Section 3 brings forward data complexity measures as they are at the core of our proposal. Section 4 describes the proposed pool generation method and its variants. Section 5 presents the experimental protocol used and discusses the observed results. Finally, Section 6 concludes this work and delineates future works.

2. Literature review

Even though there is no clear correlation between diversity and accuracy [12], it is clear that having multiple classifiers that cast votes equally is not beneficial as a single learner would suffice. Thus, some well-known methods in the literature explore diversity to create homogeneous ensembles given a base inducer. Such successful strategies have in common the manipulation of the problem data. For instance, Bootstrap Aggregating (Bagging) [4] and Boosting [5], do it horizontally since they train the pool members on different data subsets that are sampled from the training dataset.

Bagging selects instances of the classification problem at random to create training subsets via random sampling with replacement. It is a framework that performs generation and classification. Data subsets are built randomly in the generation phase, with the instances available in the training set. Each data subset results in a classifier. In the classification phase, the trained classifiers are combined with the majority voting rule [4].

Adaptive Boosting (AdaBoost) [5] is an algorithm that constructs the classifiers in a logical sequence considering the mistake of the preceding ones. It is similar to Bagging, except that instances incorrectly classified by the already generated classifiers will be more likely to participate in the training of the next classifier at each iteration. For this purpose, the sampling process considers a weight associated with each training instance from the previous iteration. At each iteration, the weight is incremented when classifiers from previous rounds misclassify an instance and otherwise decremented. The weight is used in the sampling process so that instances with higher weights are more likely to be selected for training. Consequently, the method prioritizes the most difficult instances to be used during the pool's construction.

In opposition to Bagging and Boosting, Random Subspaces (RS) [6] performs vertical data partitioning since each of the pool members is trained with a randomly selected subset of features from the original feature space. Finally, another interesting method for pool generation is the well-known Random Forest (RF) [7]. RF combines horizontal and vertical data manipulation, training trees on subsets of data while the assessment of features in the creation of split nodes is also randomized.

Different studies [13–16] have shown how data complexity metrics improve the creation of multiple classifier systems. For instance, the authors in [14] use data complexity measures in a method to define the competence regions for learning classifiers. The data complexity metrics allow them to identify the area of competence of heterogeneous classifiers, improving the performance of dynamic selection methods. Closely related to our approach, the authors in [15,16] developed a framework for dynamic selection of classifiers oriented by the difficulty of the classification problem, named DSOC. Their idea was to select classifiers based on the complexity of the classification problem. DSOC describes the use of complexity measures to generate a pool of classifiers based on two specific metrics (F1 and N2) and an optimization process using a single objective genetic algorithm. However, their focus was on a new dynamic classifier selection strategy. The most promising classifier for a given test instance is selected considering accuracy and features based on complexity measures. Such meta-features provide the similarity between the test instance neighborhood and the data subset used to train each classifier in terms of difficulty level. The motivation was to select a classifier trained on a data subset showing similar difficulty observed in the test neighborhood.

The pool generation method proposed in this paper is inspired by the framework described in [16]. Our method may represent a new first stage in that framework, where the novelties are as follows: (a) the strategy used to explore the data complexity space and also in classifier's decision space; (b) the definition of the complexity measures to be used dependent on the classification problem at hand; (c) the use of a multi-objective genetic algorithm to explore diversity in the problem complexity (two measures) simultaneously and also in the classifier decision space; (d) the strategies used to provide the final pool that can prioritize maximum diversity or maximum accuracy; and (e) the possibility of defining the inducer.

3. Data complexity measures

Data complexity measures estimate how difficult a dataset is to be categorized [8]. There are several ways to measure the complexity of classification problems. Authors in [17] categorized complexity measures into six families or groups: Overlapping, Linearity, Neighborhood,

Network, Dimensionality, and Class Balance. Within each family, there are several approaches to quantify complexity.

The remaining of this section describes the overlapping and neighborhood data complexity families. Our method uses only the measures of these two families, and their choice is justified in Section 4.1.

3.1. Overlapping measures

Overlapping measures analyze the feature space of a problem, i.e., how much the attributes of different classes are overlapping. The acronyms that name this family's metrics are: F1, F1v, F2, F3, and F4. The measures in this family are described below, and all of them require numeric attributes.

3.1.1. Maximum Fisher's Discriminant Ratio (F1)

Maximum Fisher's Discriminant Ratio (F1) is responsible for finding the distance between the centroids of two distinct classes and is based on the mean and standard deviation of each attribute [18].

Traditional F1 is applied to binary classification datasets. However, the authors in [19] proposed an equation where it is possible to calculate r_{f_j} for datasets with multiple classes. Eq. (1) demonstrates how to calculate r_{f_j}, where n is the number of examples in the class y_j, $\mu_{y_j}^{f_i}$ denotes the average of the attribute f_i over the examples of the class y_j, the μ^{f_i} is the average of each characteristic f_i across all classes of the problem. Finally, $x_{l_i}^{j}$ represents the individual value (l) of the attribute f_i for an example of the y_jth class.

$$r_{f_i} = \frac{\sum_{j=1}^{n} n_j(\mu_{y_j}^{f_i} - \mu^{f_i})^2}{\sum_{j=1}^{n}\sum_{l=1}^{y_j}(x_{l_i}^{j} - \mu_{y_j}^{f_i})^2} \tag{1}$$

3.1.2. The directional-vector maximum Fisher's Discriminant Ratio (F1v)

The F1v measure determines the projection vector that separates two classes according to the Fisher equation. Malina [20] analyzed different hyperplanes of binary and non-binary problems proposing Eq. (2) to find the distance between the centroids of other classes.

$$F1v = \frac{d^S B d}{d^S W d} \tag{2}$$

In Eq. (2), d is a directional vector that stores data that is designed to maximize the spread of classes, S is a subproblem, B is the matrix of attributes between classes, and W is an array of attributes in the label itself.

3.1.3. Volume of overlap region (F2)

According to Ho and Basu [13], and Lorena et al. [17], F2 measures the distance and checks how overlapping are the values of an attribute between across classes. F2 accounts for the minimum and maximum of each attribute of the same label. The overlapping interval is calculated by normalizing the attributes of both classes and multiplying them. The determination of F2 is obtained by Eq. (3) that is the division of $overlap(f_j)$ and $range(f_j)$ components computed for each feature f_j available in the dataset.

$$F2 = \prod_j \frac{\text{overlap}(f_j)}{\text{range}(f_j)} = \prod_j \frac{\max\left\{0, \min\max(f_j) - \max\min(f_j)\right\}}{\max\max(f_j) - \min\min(f_j)} \tag{3}$$

In the equation above, $\min\max(f_j) = \min(\max(f_j^{y_1}), \max(f_j^{y_2}))$, $\max\min(f_j) = \max(\min(f_j^{y_1}), \min(f_j^{y_2}))$, $\max\max(f_j) = \max(\max(f_j^{y_1}), \max(f_j^{y_2}))$, and $\min\min(f_j) = \min(\min(f_j^{y_1}), \min(f_j^{y_2}))$.

3.1.4. Feature efficiency (F3)

F3 checks whether there is an overlap between different classes per attribute and not the entire feature set. If there is overlap, the targets are considered ambiguous in this region. Thus, the percentage that each characteristic contributes to the separation of two classes results in the F3 measure. According to Orriols et al. [18], the calculation of F3 is done following the heuristic: for each attribute, an overlap region is considered (where the attribute value is the same or very close for different classes), the result of F3 will be the ratio of the number of instances in the set that are not in this region, over the total number of examples. The maximum value found will be the value of F3.

3.1.5. Collective feature efficiency (F4)

The F4 measure uses a more discriminating set of attributes of each class. F4's heuristic consists of (i) separating all the examples that an attribute can distinguish; (ii) from the remaining examples, there is the attribute that most distinguishes the classes of the problem; (iii) the previous two steps are repeated until all the examples have been classified or there are no more dataset attributes. Finally, F4 is the percentage of examples that were discriminated by the attributes [17,18].

3.2. Neighborhood measures

Neighborhood measures define the distance between instances and or attributes to determine the border regions of classes and the difficulty in separating them. The measures of this family only accept problems with numerical attributes. This group includes N1, N2, N3, N4, T1, and LSCAvg measures.

3.2.1. Fraction of borderline points (N1)

N1 defines the percentage of examples that are close to the boundary line between different problem labels. N1 applies Minimum Spanning Tree (MST) [21]. This algorithm generates a graph connecting all the set elements so that the sum of the connections is as tiny as possible. The edges of this tree are connected in two different classes representing the examples that are in the boundary line [13,17,22].

Eq. (4) defines the sum of the edges that are in the boundary region, n denotes the number of examples of the problem, x_i, and x_j are instances of the problem, b is an indicator of function, just like in measure F3, it represents values between zero and one. The variables y_i and y_j determine the labels for each instance.

$$N1 = \frac{1}{n}\sum_{i=1}^{n} b\left((x_i, x_j) \in \text{MST} \wedge y_i \neq y_j\right) \tag{4}$$

3.2.2. Ratio of intra/extra class nearest neighbor distance (N2)

The N2 descriptor evaluates the intersection of two classes through the distance between examples of different targets of the problem. N2 is summed up in the sum of the length of the nearest neighbor of an instance of the same label by the sum of another neighbor, but with another target [13,17,23]. These steps are performed for all instances of the problem.

The Eq. (5) refers to N2, where x_i represents the examples i, $intraDist$ represents the distance from the neighbor of x_i being of the same class, and $interDist$ represents the distance from nearest neighbor x_i, where class is different. Finally, n represents the total number of problem instances and labels.

$$N2 = \frac{\sum_{i=0}^{n} intraDist(x_i)}{\sum_{i=0}^{n} interDist(x_i)} \tag{5}$$

3.2.3. Error rate of the nearest neighbor classifier (N3)

The nearest neighbor classifier error rate refers to the 1NN classifier error rate estimated by *leave-one-out*, for a neighbor of the test instance [17,19,34]. N3 is measured according to Eq. (6), where NN is the closest neighbor of the classifier prediction for example x_i using all training instances [17].

$$N3 = \frac{\sum_{i=1}^{n} b(NN(x_i) \neq y_i)}{n} \tag{6}$$

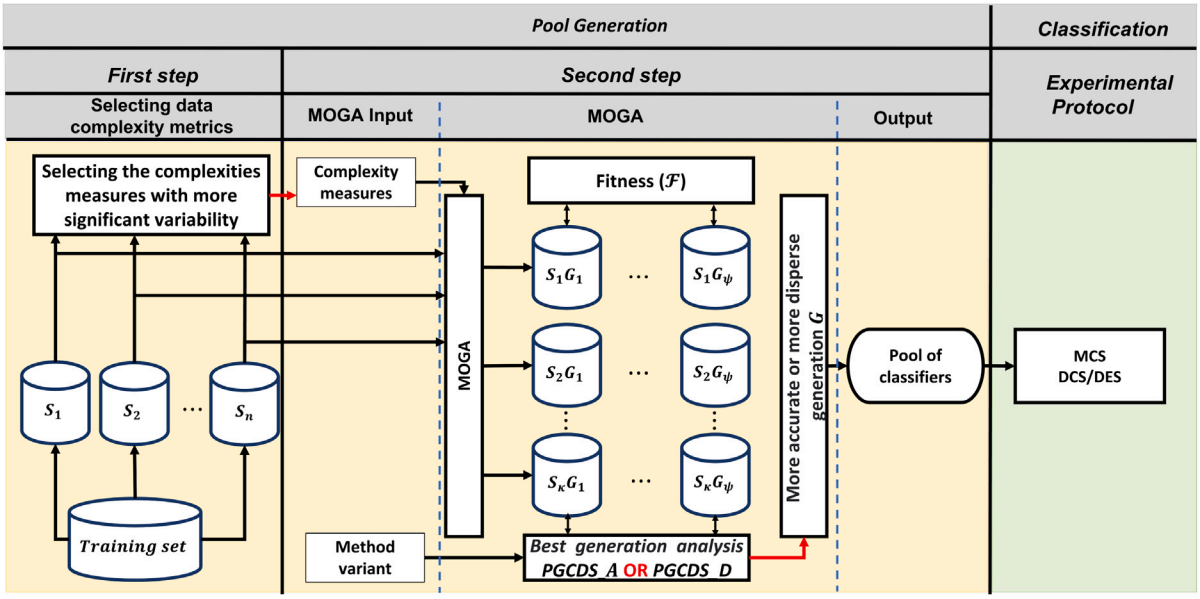


Fig. 1. Method variants based on Maximum Accuracy and Maximum Dispersion. G_r represents the pool generation.

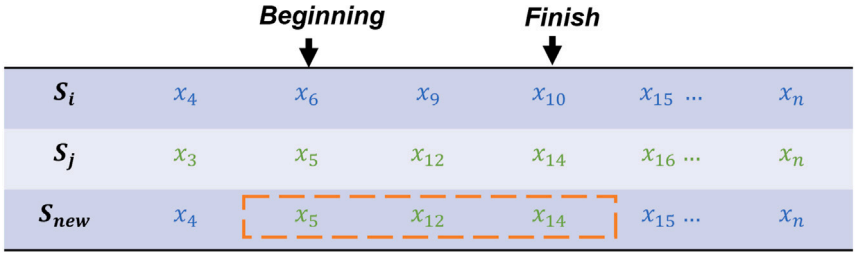


Fig. 2. Crossover process starts with two bags (S_i and S_j) drawn in the population. The instances in S_i are swapped with those in S_j in the `beginning` and `finish` range.

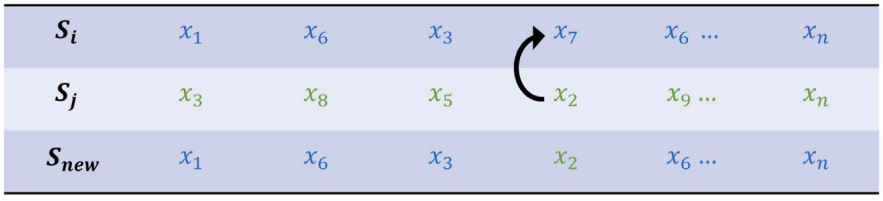


Fig. 3. Mutation process starts with two bags (S_i and S_j) drawn from the population. S_i has instance x_7 replaced by x_2 from S_j.

Table 1
Methods from the literature and the parameters used.

Method	Parameters	Library/version	Ref.
Perceptron	max_iter=100, tol=1.0	scikit-learn 0.24.2	[24]
Decision tree	All default	scikit-learn 0.24.2	[24]
BaggingClassifier	n_estimators=100, max_sample=0.5, random_state=86	scikit-learn 0.24.2	[24]
AdaBoostClassifier	n_estimators=100, random_state=86	scikit-learn 0.24.2	[24]
RandomForestClassifier	n_estimators=100, random_state=86	scikit-learn 0.24.2	[24]
DCS Methods	All defaults	DESlib 0.3	[25]
DES Methods	All defaults	DESlib 0.3	[25]
NSGA-II	All defaults	deap 1.3.1	[26]
Complexities measures	All defaults	ECoL 0.3.0	[17]

3.2.4. Non-linearity of the nearest neighbor classifier (N4)

Measure N4 returns the error rate of the KNN classifier. The N4 metric uses a synthetic problem created from a training set to create a synthetic test subproblem. The artificial set is made through interpolation between randomly chosen pairs within the same class [13, 18]. Interpolation randomly pulls two examples from the training set of the same category. After that, a synthetic example is created between the two instances. The process is repeated until all the examples are combined. The KNN algorithm will classify the cases made by interpolation.

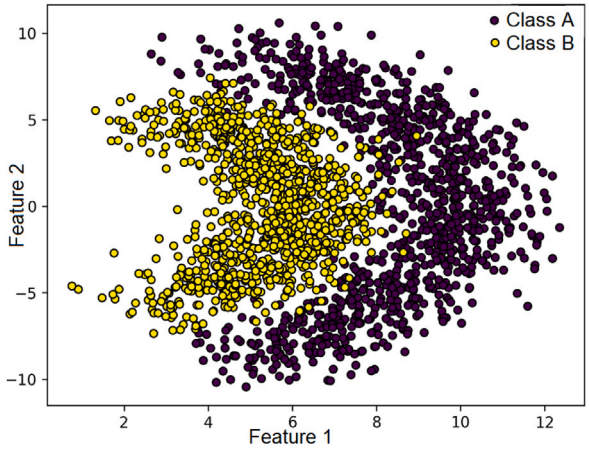


Fig. 4. Representation of Lithuanian Dataset.

Table 2
Datasets used in the experiment.

No.	Datasets	Instances	Features	Classes	Source
01	Australian	690	14	2	UCI [27]
02	Banana	2000	2	2	PRTools [28]
03	Blood	748	4	2	UCI [27]
04	CTG	2126	21	3	UCI [27]
05	Diabetes	766	8	2	UCI [27]
06	Faults	1941	27	7	UCI [27]
07	German	1000	24	2	STATLOG [29]
08	Haberman	306	3	2	UCI [27]
09	Heart	270	10	2	STATLOG [29]
10	ILPD	583	34	2	UCI [27]
11	Ionosphere	351	16	2	UCI [27]
12	Laryngeal1	213	16	2	LKC [30]
13	Laryngeal3	353	2	3	LKC [30]
14	Lithuanian	2000	2	2	PRTools [28]
15	Liver	345	6	2	UCI [27]
16	Mammo	830	5	2	KEEL [31]
17	Monk	432	6	2	KEEL [31]
18	Phoneme	5404	5	2	ELENA [32]
19	P2	5000	2	2	[33]
20	Segmentation	2310	19	7	UCI [27]
21	Sonar	208	60	2	UCI [27]
22	Thyroid	692	16	2	LKC [30]
23	Vehicle	846	18	4	STATLOG [29]
24	Vertebral	300	6	2	UCI [27]
25	WBC	569	30	2	UCI [27]
26	WDVG	5000	21	3	UCI [27]
27	Weaning	302	17	2	LKC [30]
28	Wine	178	13	3	UCI [27]

Eq. (7) defines N4, where Inp represents the number of interpolations performed, b the result of the function indicator, $resKNN$ represents the prediction of the KNN algorithm for the synthetic instance x'_i with label y'_i [17].

$$N4 = \frac{1}{Inp}\sum_{i=1}^{Inp} b(resKNN(x'_i) \neq y'_i) \tag{7}$$

3.2.5. Fraction of hyper-spheres covering data (T1)

T1 consists of creating spheres, the center of which is in an instance of a class. The selection of the initial instance is random. The spheres' radius are increased until some instance of another class touches its edge. The final T1 value is given by dividing the number of spheres required to cover a class divided by the total number of instances in the problem [8,13,17].

3.2.6. The local set average cardinality measure (LSCAvg)

According to Leyva, González, and Perez [35], the local set of an instance is defined as its neighbors within a radius that is smaller than the distance from this sample to a sample from a different class. The cardinality of a local set (LS) indicates the proximity of an instance to the decision boundary. Therefore, LS cardinality is lower for instances separated from the other class. This metric represents the average LS value obtained for all instances in a dataset.

4. Proposed method

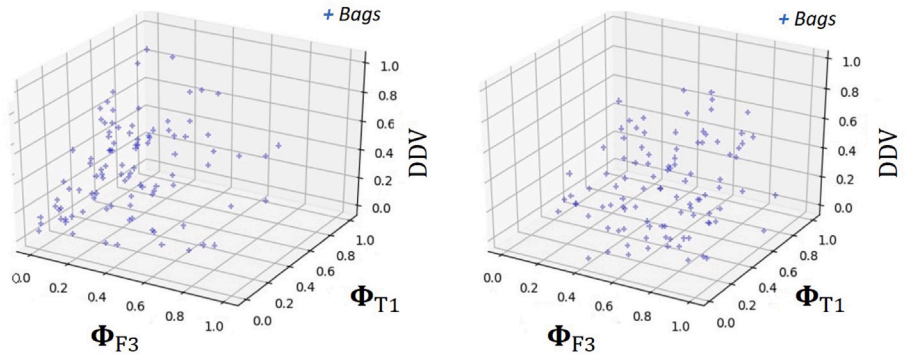
Countless works in the literature seek to improve MCS performance by generating diverse classifiers. The novelty in our approach is to drive the pool training on subproblems of the original classification problem that are represented by subsets of data created using a sampling mechanism based on complexity measures. More than contributing to the entire pool performance, we expect that the classifiers trained in that fashion increase the probability of dynamic selection algorithms choosing the classifier(s) with more competence for an input instance.

The proposed method, hereafter referred to as *Classifier Pool Generation based on Diversity in the Decision and Complexity Spaces*, or PGDCS, generates a homogeneous pool of classifiers. It has two main steps: (i) selecting data complexity metrics and (ii) pool generation using multi-objective genetic algorithms (MOGAs). More specifically, the first step consists of selecting the complexity measures with the higher variability among n data subsets randomly created from a given classification problem. The rationale is to choose complexity measures that show an extensive variance range. After defining the most suitable complexity measures for the given problem, in the second step of the method, such metrics are used with a measure of diversity in the classifiers' decision space as fitness functions of a multi-objective genetic algorithm (MOGA) to create data subsets (or subproblems) in a two-level diversity strategy. Finally, we performed the experiments using dynamic classifier selection methods. The MOGA provides subsets that better cover the data complexity space while generating a diverse pool in classifier decisions. The final pool of classifiers is obtained according to the generation that presents the best performance (if the accuracy variant is chosen) or the generation showing the highest average distance between individuals (if the diversity variant is selected). After performing the two steps of the proposed method, the pool of classifiers generated can be used in multiple classifier systems (MCS) with or without considering a classifier selection stage. In our experimental protocol (Section 5), we have performed experiments by combining all pool members and also by using DCS and DES methods. We expect an important contribution when using the proposed pool generator for DCS and DES methods since the literature shows that their success usually depends on a pool of diverse and accurate classifiers. Fig. 1 overviews the proposed method which is detailed in the next sections.

4.1. Selecting data complexity metrics

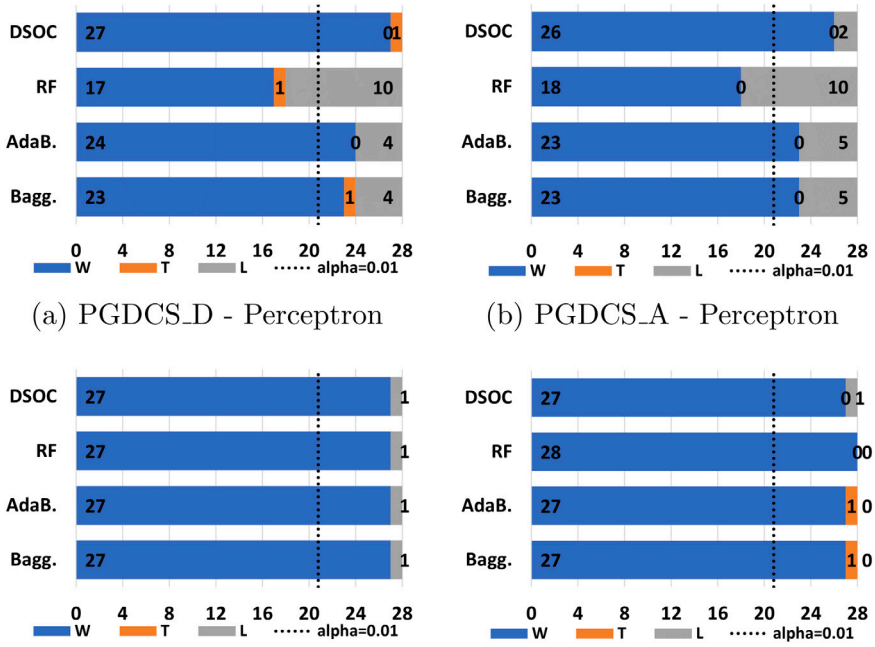
Even though our proposal is generic, the complexity measures used in the proposed method belong to the Overlapping and Neighborhood families as their computational cost is smaller and yielded interesting results in previous studies in multiple classifier systems [16]. We have considered five overlapping-based metrics, named $\omega_1 = \{F1, F1v, F2, F3, F4\}$, and six neighborhood-based ones, named $\omega_2 = \{N1, N2, N3, N4, T1, LSC\}$.

The original problem dataset is first divided into training, validation, and testing sets. Next, at each iteration $t < T$ of the proposed algorithm, we randomly generate n data subsets ($S^t_n = S^t_1, S^t_2, \ldots, S^t_n$) from the training set with replacement such that each subset contains u percent of the training instances available. Similar subsampling is performed in the Bagging algorithm. The two families of measures (ω_1 and ω_2) are then computed for each S^t_i. The value of each metric (α^t_{ij}) is normalized across the n data subsets using the min–max normalization as denoted in Eq. (8), where $\alpha^t_{min,j}$ and $\alpha^t_{max,j}$ represent the lowest and



(a) First generation, whose average distance is 0.4071 (b) Best generation, whose average distance is 0.453

Fig. 5. Representation of the dispersion of individuals in the Lithuanian database during the execution of the MOGA. Fig. (a) demonstrates the dispersion in the first generation of individuals and Fig. (b) represents the best generation of PGDCS.



(a) PGDCS_D - Perceptron (b) PGDCS_A - Perceptron

(c) PGDCS_D - Decison tree (d) PGDCS_A - Decison tree

Fig. 6. Win, Tie and Loss of the method variants (PGDCS_A, PGDCS_D) when compared to Bagging, AdaBoost, Random Forest (RF), and DSOC. Perceptron and Decision Tree as base classifiers. aggregation based on MVR.

Table 3
Average dispersion (votes) in 10 interactions of the Lithuanian Problem.

#	Family of measure										
Iteration	Overlapping					Neighborhood					
	F1	F1v	F2	F3	F4	N1	N2	N3	N4	T1	LSC
1	0.194	0.181	0.219	**0.234**	0.233	0.197	0.208	0.183	0.190	0.197	0.181
2	0.220	0.207	0.229	0.228	**0.236**	0.215	0.215	**0.242**	0.163	0.174	0.183
3	0.230	0.210	0.197	**0.238**	0.214	0.173	0.201	**0.215**	0.214	0.210	0.172
4	0.189	0.186	0.225	0.232	**0.237**	0.203	0.185	0.184	0.217	**0.221**	0.185
5	0.230	0.229	0.211	0.216	**0.248**	0.205	0.168	0.236	0.183	**0.245**	0.205
6	**0.243**	0.234	0.211	0.222	0.223	0.170	0.173	0.228	0.227	**0.246**	0.220
7	0.189	0.184	0.184	**0.206**	0.203	**0.217**	0.214	0.195	0.205	0.188	0.179
8	0.193	0.187	**0.204**	0.187	0.184	0.189	**0.220**	0.214	0.184	0.203	0.200
9	0.205	0.207	0.216	**0.238**	0.208	0.201	0.202	0.206	0.199	0.205	**0.211**
10	0.180	0.176	**0.225**	0.220	0.206	**0.221**	0.201	0.193	0.166	0.167	0.204
Votes	1	0	2	**4**	3	2	2	2	0	**3**	1

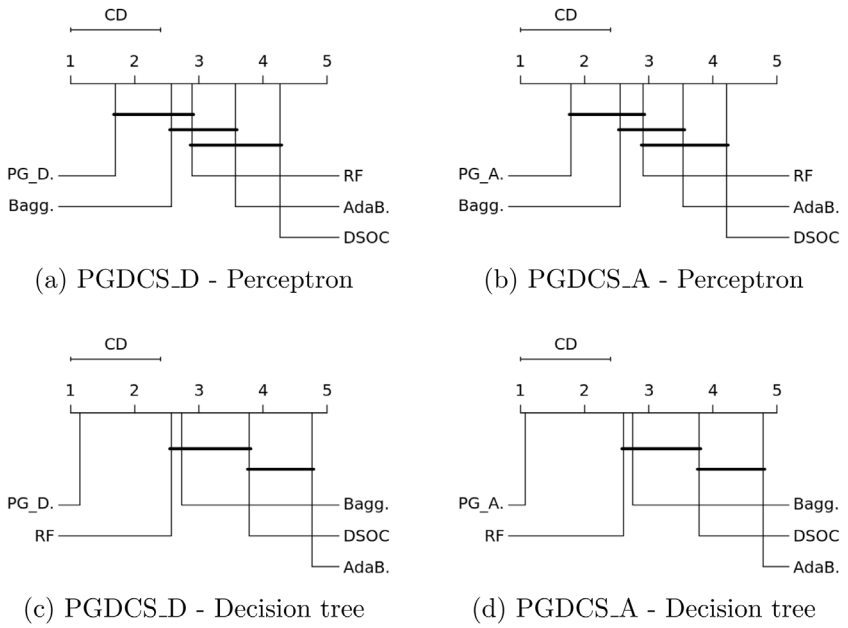


Fig. 7. Nemenyi analysis: (PGDCS_A, PGDCS_D) compared to Bagging, AdaBoost, Random Forest (RF), and DSOC. Perceptron and Decision Tree as base classifiers. aggregation based on MVR.

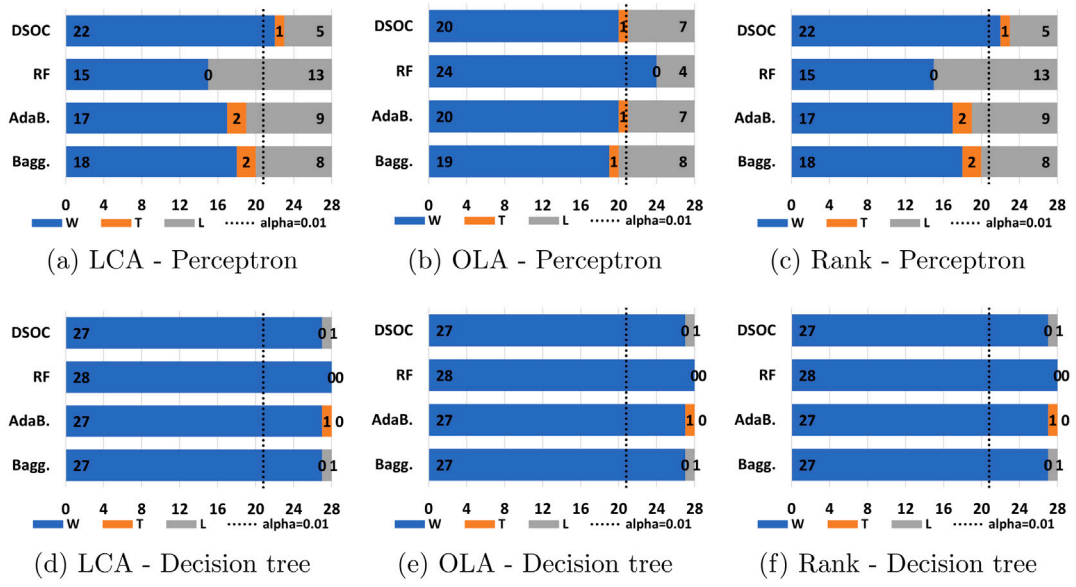


Fig. 8. PGDCS_A variant considering both base inducers (Perceptron and Decision Tree) compared to other pool generators when considering DCS methods.

highest value of the jth complexity measure during the tth iteration, respectively.

$$\alpha_{ij}^{t} = \frac{\alpha_{ij}^{t} - \alpha_{min,j}^{t}}{\alpha_{max,j}^{t} - \alpha_{min,j}^{t}} \tag{8}$$

After normalization, the algorithm computes the dispersion for each complexity measure using Eq. (9), where μ_{tj} represents the average of the jth complexity measure at iteration t, and n is the number of data subsets.

$$\sigma_{j}^{t} = \sqrt{\frac{\sum_{i=1}^{n}(\alpha_{i,j}^{t} - \mu_{j}^{t})^{2}}{n}} \tag{9}$$

The complexity measure with the highest dispersion value inside of each family (ind) is obtained as described in Eq. (10).

$$\text{ind} = \underset{j \in \omega_k}{\text{argmax}}\, \sigma_{j}^{t} \tag{10}$$

The rationale behind selecting the most disperse metric is to have more flexibility in the second step when we target sub-problems with different difficulty levels. In Eq. (11), the complexity measure showing a large dispersion receives a vote at each iteration t.

$$\text{votes}_{\text{ind}}^{\omega_k} = \text{votes}_{\text{ind}}^{\omega_k} + 1 \tag{11}$$

The metric most voted of each family (cm_1 and cm_2) is selected for the classification problem analyzed as denoted in Eq. (12). If a tie occurs, the metric with the highest average dispersion across the T iterations is selected. As a result of this process, the two selected complexity metrics, one per family, are part of the objective functions used during the next step of the proposed method.

$$\text{cm}_k = \underset{j \in \omega_k}{\text{argmax}}\, \text{votes}_{j}^{\omega_k} \tag{12}$$

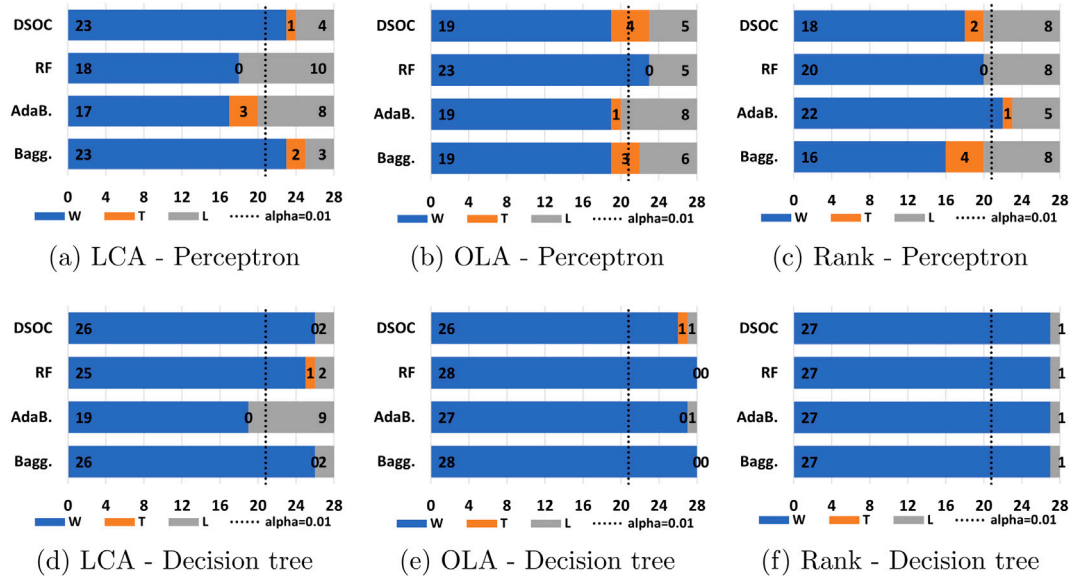


Fig. 9. PGDCS_D variant considering both base inducers (Perceptron and Decision Tree) compared to other pool generators when considering DCS methods.

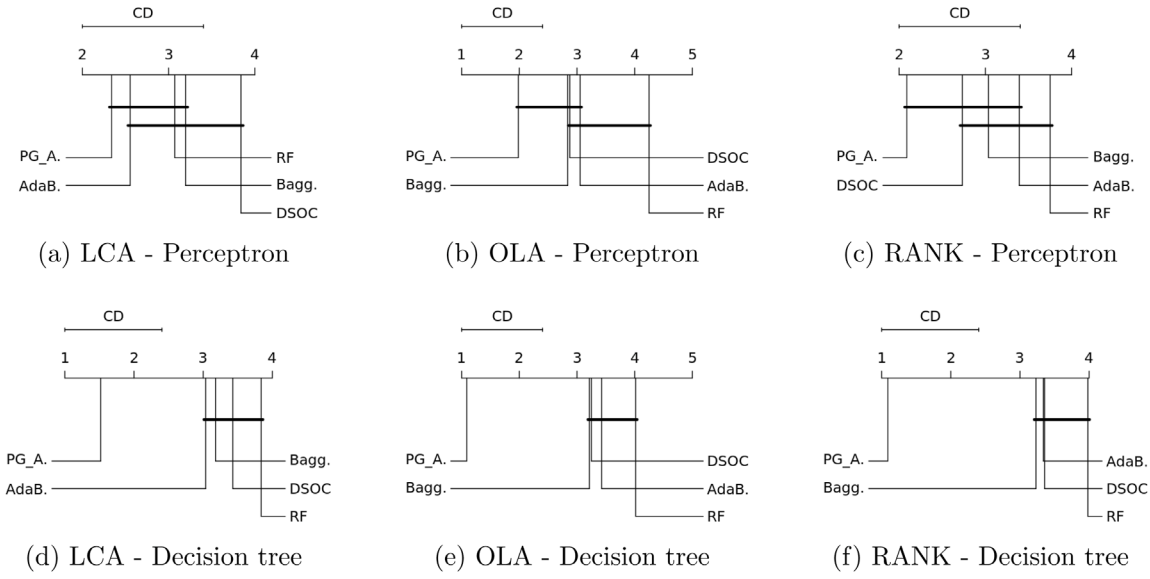


Fig. 10. Nemenyi analysis comparing PGDCS_A (PG_A) to other pool generation methods in different DCS approaches.

4.2. Pool generation based on multi-objective optimization

In the second stage, a multi-objective genetic algorithm (MOGA) [10] uses the measures of complexity previously selected for the problem at hand and a diversity measure to generate a pool of classifiers. The idea is to explore diversity in the data complexity and classifier decision spaces simultaneously. To obtain the best pool, we evaluated two strategies. The first outputs the pool related to the generation with the best accuracy (*maximum accuracy*), named PGDCS_A. The second, named PGDCS_D, outputs the pool of the generation with the most significant average distance between individuals (*maximum diversity*). The rationale behind this second strategy is to focus on variety, providing a pool with members highly disperse in both input and output spaces. With the proposed method, we expect to obtain a pool competitive in terms of accuracy compared to traditional methods and improve MCS based on dynamic selection.

4.2.1. MOGA

Algorithm 1 presents the multi-objective genetic algorithm used. It receives as input the number of generations ψ, the population size κ, the number of new individuals after crossover and mutation θ, the training set Tr, the validation set V, the base estimator I, and Op as the method variant. At $t = 0$, the first population of data subsets $P(t)$ is created with size κ in line 3. In line 4, $F(t)$ receives the three fitness values of each individual of the generation t, which are computed using Algorithm 2. The loop in lines 5–30 depicts the iterations of the genetic algorithm. Line 8 depicts the crossover and mutation steps, which are followed by the offspring definition in line 10. Next, line 13 to 27, in case PGDCS_A variant, the global population accuracy υ on the validation dataset V is computed using majority voting rule. In contrast, the dispersion-guided variant (PGDCS_D) calculates the average dispersion of the pool members using Eq. (13). To that end, for each generation we keep a global descriptor F which is composed of three objective values: Φ_{cm_1}, Φ_{cm_2}, and $DDV[C_i]$. The best solution in the PGDCS_D variant is defined considering the generation that has the

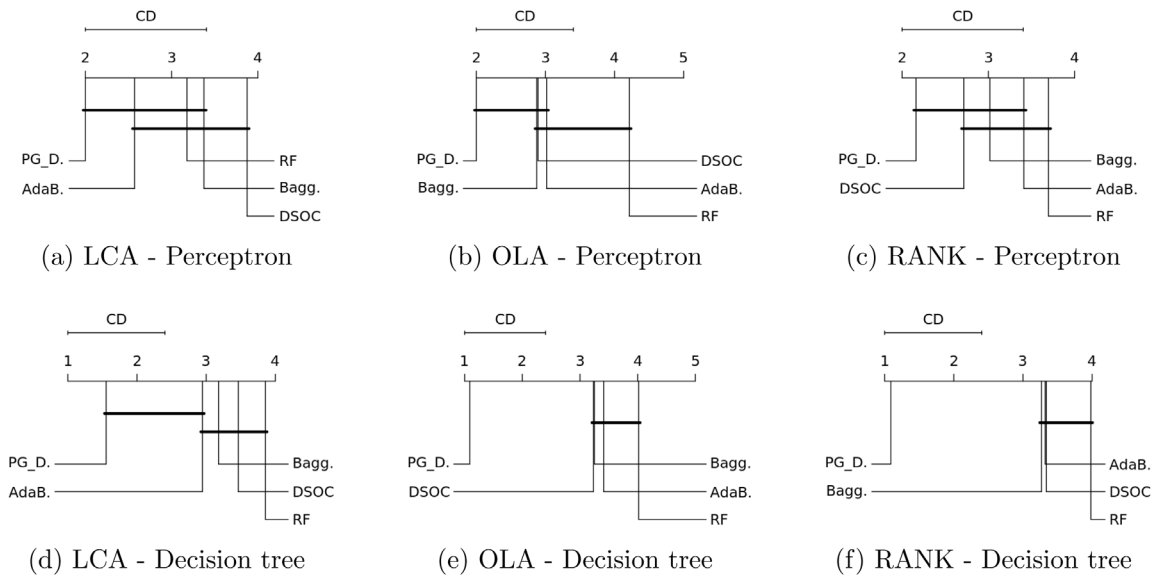


Fig. 11. Nemenyi analysis comparing PGDCS_D (PG_D) to other pool generation methods in different DCS approaches.

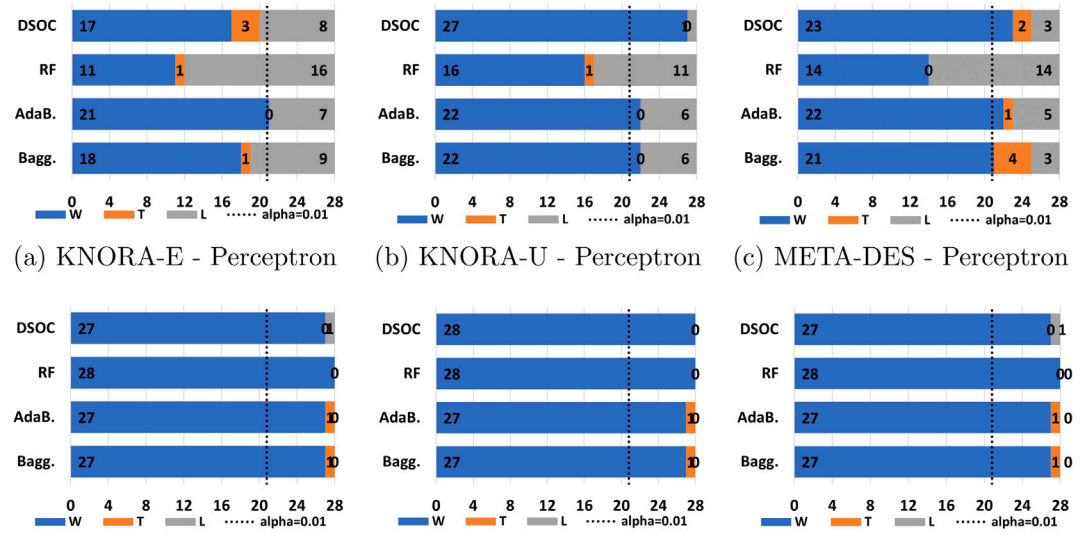


Fig. 12. Win, tie and loss of the PGDCS_A variant against to other pool generators when considering DES methods.

highest dispersion considering the F descriptor computed in Eq. (13), where $G_{disp}(t)$ is the global dispersion at generation t, κ is the number of bags/classifiers or population size and F_n is the number of objective functions.

$$G_{disp}(t) = \frac{\sum_{i,j=1}^{\kappa} \sqrt{\sum_{h=1}^{F_n} (F[i,h] - F[j,h])^2}}{\kappa - 1} \quad (i,j = 1..\kappa)(i \neq j) \tag{13}$$

In such a variant, we hypothesize that the population with the highest global dispersion ϱ yields data subsets that better cover the problem complexity space and consequently a pool trained on such data subsets has increased diversity in the decision (output) space.

It is worth noting that during the evolution of generations, the MOGA uses the three described fitness and the Pareto front strategy. We have computed the population average accuracy or diversity dispersion for each generation to select the best one according to the method variant used, PGDCS_A or PGDCS_D, respectively. It means that the best pool is not always related to the last MOGA generation.

When comparing with well-known ensemble learning methods available in the literature, we can see that the proposed method is time consuming. However, even paying this price, the experimental results have shown that exploring diversity in both spaces (data complexity and classifier decision) can significantly improve accuracy for most of the classification problems evaluated in Section 5. Remarkably, this alternative for ensemble generation gave the dynamic selection methods a greater precision according to our experiments.

4.2.2. Fitness function

The optimization process occurs based on three objectives. The first two explore the data complexity space using the metrics selected in the first phase of the method. The idea is to maximize the average distance (dispersion) among the data subsets considering each complexity metric computed in a pairwise manner. The third objective is to maximize the diversity in the decision space by using the Double Fault (DF) metric computed on the decisions of the classifiers trained on each data subset also in a pairwise manner. The adoption of the double fault metric follows the findings of [9], where superior results in majority vote ensembles were obtained via DF optimization.

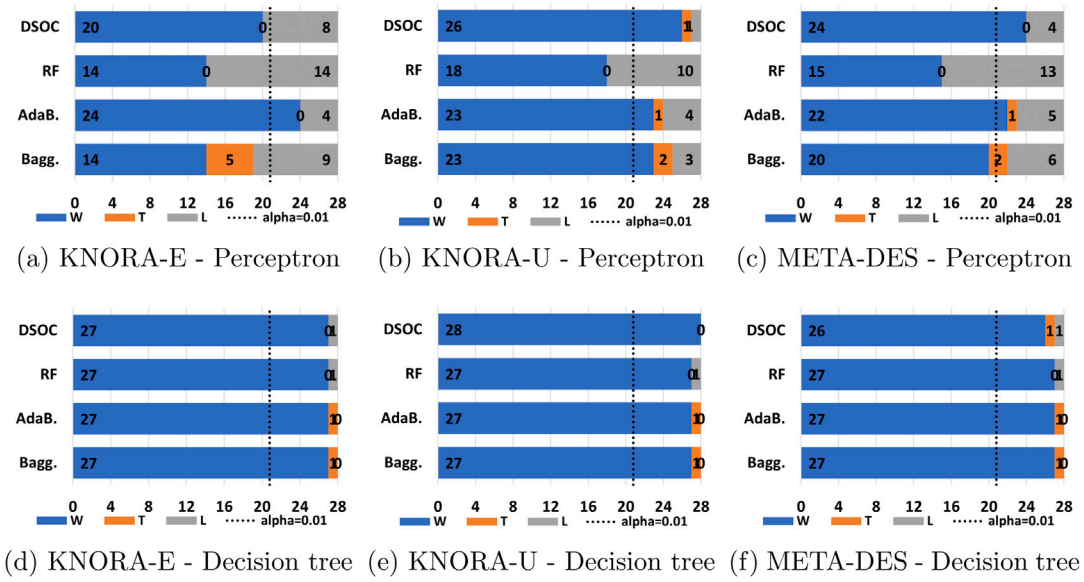


Fig. 13. Win, tie and loss of the PGDCS_D variant against to other pool generators when considering DES methods.

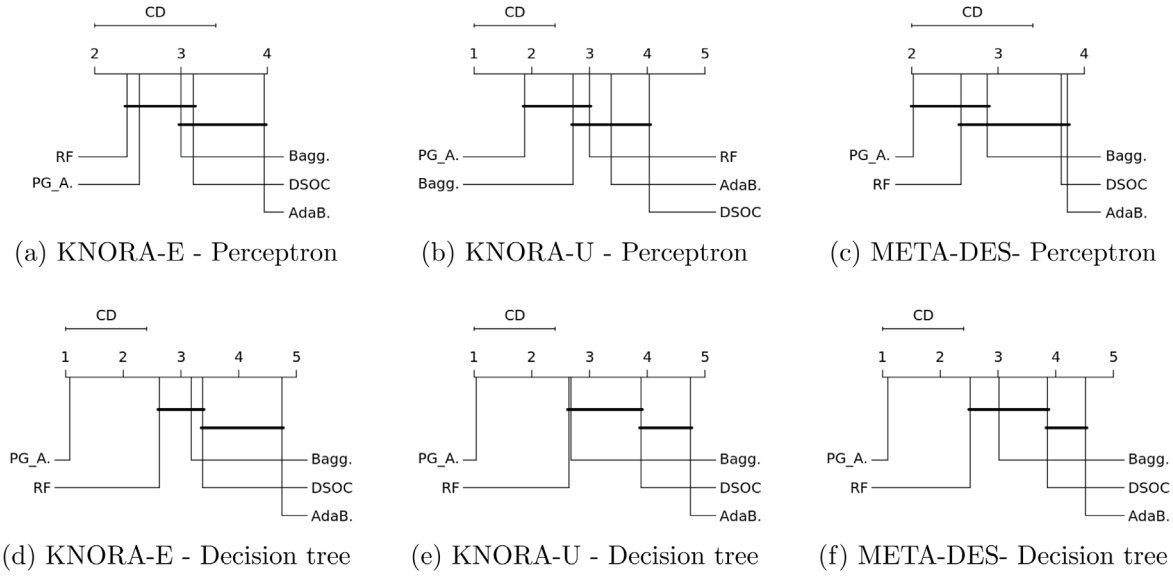


Fig. 14. Nemenyi analysis comparing PGDCS_A (PG_A) to other pool generation methods in different DES approaches.

The objectives are computed considering each data subset S_i, as follows:

- $\Phi_{cm_1}[S_i]$ and $\Phi_{cm_2}[S_i]$: the dispersion values of the subset S_i considering the cm_1 and cm_2 complexity measures selected in a problem-specific fashion (see Section 4.1). To obtain the dispersion, the algorithm starts by computing $\varphi_{cm_j}[S_i]$, which is the value of the metric cm_j for each individual in the population, a subset of data S_i. Next, in a pairwise manner, the difference between S_i and all other data subset complexity values $\varphi_{cm_j}[S_k]$ is computed as denoted in Eq. (14), where κ is the total number of subsets in the population (population size). The dispersion $\Phi_{cm_j}[S_i]$ stores the average distance of the subset S_i with respect to all other subsets in the complexity space regarding cm_j. The idea is to maximize these dispersion values to better cover the problem complexity space.

$$\Phi_{cm_j}[S_i] = \frac{\sum_{k=1}^{\kappa} abs(\varphi_{cm_j}[S_i] - \varphi_{cm_j}[S_k])}{\kappa - 1} \tag{14}$$

- $DDV[C_i]$ is the Decision-based Diversity Value of a classifier C_i computed using the Double Fault diversity measure. The idea is to maximize the diversity w.r.t. classifiers decisions. First, a classifier (C_i) is trained for each subset of data (S_i) representing a subproblem with a given complexity level. Next, the algorithm calculates ($DF(C_i, C_j)$), which is the Double Fault diversity measure per classifier using the validation set. Finally, a final diversity value is estimated for S_i in a pairwise manner as denoted in Eq. (15), where κ, the population size, here represents the number of classifiers (or data subsets). Besides, DF is a score in the interval $[0, 1]$ obtained as a fraction of the number of instances that have been misclassified by both classifiers C_i and C_j, denoted in Eq. (16) as e, by the total number of samples in a validation set V.

$$DDV(C_i) = \frac{\sum_{j=1, j!=i}^{\kappa} DF(C_i, C_j)}{\kappa - 1} \tag{15}$$

$$DF = \frac{e}{|V|} \tag{16}$$

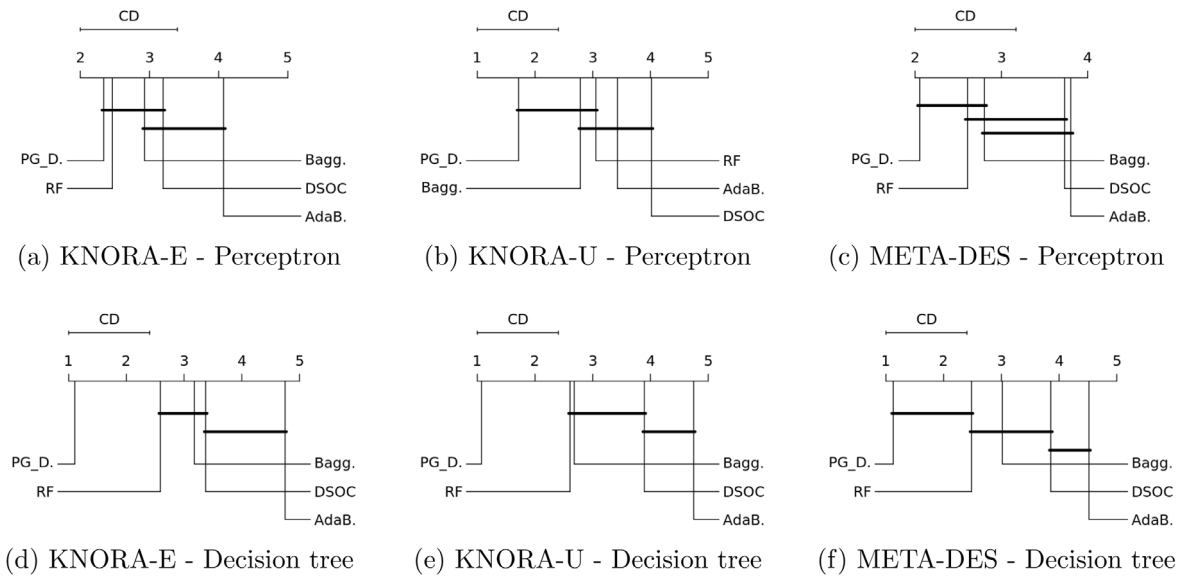


Fig. 15. Nemenyi analysis comparing PGDCS_D (PG_D) to other pool generation methods in different DES approaches.

Table 4
Voting results for measures with highest standard deviation.

Dataset	F1	F1v	F2	F3	F4	N1	N2	N3	N4	T1	LSC
Australian	**4**	3	1	2	0	0	0	2	**5**	1	2
Banana	0	2	3	**4**	1	1	0	3	0	**4**	2
Blood	2	0	**5**	3	0	**3**	1	1	2	1	2
CTG	**4**	3	2	1	0	1	**3**	2	1	2	1
Diabetes	1	1	2	**5**	1	1	3	1	0	1	**4**
Faults	0	**10**	0	0	0	1	1	**4**	1	1	2
German	0	0	**9**	1	0	3	1	0	**4**	1	1
Haberman	2	2	**4**	0	2	0	1	3	**4**	1	1
Heart	0	0	0	0	**10**	0	0	0	**6**	3	1
ILPD	1	3	0	**5**	1	**4**	2	2	1	0	1
Ionosphere	**7**	2	0	1	0	0	1	3	**5**	1	0
Laryngeal1	1	0	0	1	**8**	0	2	2	**4**	0	2
Laryngeal3	0	3	0	2	**5**	2	**4**	2	1	0	1
Lithuanian	1	0	2	**4**	3	2	2	2	0	**3**	1
Liver	1	5	2	0	2	2	1	**3**	2	2	0
Mammo	1	2	**7**	0	0	2	1	1	**3**	2	1
Monk	0	0	4	0	**6**	1	**4**	1	2	2	0
P2	2	**5**	3	0	0	1	2	**4**	2	0	1
Phoneme	1	**4**	1	2	2	1	3	**4**	1	0	1
Segmentation	0	**10**	0	0	0	0	2	**4**	1	2	1
Sonar	**5**	2	0	1	2	2	0	2	**5**	1	0
Thyroid	2	0	0	**5**	3	0	2	2	1	0	**5**
Vehicle	0	0	0	1	**9**	1	0	3	0	**4**	2
Vertebral	1	2	1	0	**6**	2	**4**	2	2	0	0
WBC	3	2	0	**5**	0	1	2	0	**3**	2	2
WDVG	**5**	0	0	2	3	3	1	2	0	**4**	0
Weaning	3	3	0	**4**	0	0	2	0	**6**	0	2
Wine	1	1	0	**8**	0	4	0	**5**	0	1	0
Average	1.7	2.3	1.6	2	2.3	1.4	1.6	2.1	2.2	1.4	1.3

Table 5
MOGA parameters used in our method.

Name	Symbol	Value
Number of generations	ψ	20
Population size	κ	100
Number of children	θ	100
Offspring size	γ	100
Crossover rate (probability)	–	0.9
Mutation rate (probability)	–	0.1

Algorithm 2 describes the fitness function adopted in our method where the inputs are: the population P and the corresponding number of individuals (κ). In the first loop (lines 2 to 6), the complexity values related to the selected metrics of overlapping and neighborhood of each subset S_i are computed and stored in $\varphi_{cm_1}[S_i]$ and $\varphi_{cm_2}[S_i]$, respectively. Besides, a classifier is trained for each S_i considering a previously defined base classifier. In the second loop (lines 7 to 11), in a pairwise manner, the algorithm computes for each S_i the dispersion Φ_{cm_1} and Φ_{cm_2} related to the measures cm_1 and cm_2 using Eq. (14). In addition, the Double Fault metric for each S_i is computed using Eq. (15).

4.2.3. Crossover

One of the essential elements of a multi-objective genetic algorithm (MOGA) is the crossing operator, which is responsible for generating new individuals and creating a new population. The crossing operator used in our method randomly chooses two individuals within the

Table 6
Average generation number round chosen in MOGA per dataset.

Dataset	Perceptron		Decision Tree	
	PGCDS_A	PGCDS_D	PGCDS_A	PGCDS_D
Australian	7.6 (4.5)	12.5 (0.9)	10.5 (1.1)	14.6 (3.3)
Banana	2.9 (2.8)	6.6 (5.0)	8.0 (0.0)	7.8 (0.7)
Blood	3.5 (6.2)	9.0 (5.7)	6.5 (1.7)	11.8 (4.1)
CTG	19.1 (2.2)	10.1 (2.1)	9.2 (4.8)	11.0 (2.0)
Diabetes	11.4 (4.8)	17.5 (0.9)	12.5 (3.9)	12.0 (3.7)
Faults	6.3 (6.3)	10.6 (3.2)	9.5 (2.7)	13.6 (0.5)
German	9.9 (0.2)	16.1 (0.2)	10.6 (3.7)	12.8 (2.3)
Haberman	9.8 (1.7)	6.8 (4.1)	5.3 (0.7)	14.5 (2.2)
Heart	9.2 (2.5)	13.2 (4.9)	7.2 (3.2)	8.8 (1.6)
ILPD	13.9 (2.2)	12.8 (2.2)	13.4 (1.3)	17.4 (2.8)
Ionosphere	7.5 (2.2)	11.0 (3.9)	9.9 (3.3)	11.0 (3.9)
Laryngeal1	15.8 (2.2)	12.8 (1.3)	11.6 (4.5)	12.5 (5.0)
Laryngeal3	12.1 (1.8)	13.4 (6.7)	11.3 (3.5)	12.5 (2.3)
Lithuanian	12.8 (5.2)	15.0 (2.4)	7.5 (2.2)	11.9 (4.3)
Liver	18.6 (1.1)	6.5 (1.1)	11.4 (3.6)	11.1 (5.6)
Mammo	5.4 (4.3)	10.3 (3.1)	12.2 (1.9)	10.7 (4.0)
Monk	16.4 (0.8)	10.5 (1.5)	13.7 (0.7)	11.6 (4.4)
P2	2.0 (0.0)	11.8 (3.4)	9.0 (3.1)	10.9 (3.8)
Phoneme	8.7 (4.4)	15.5 (5.0)	11.1 (5.2)	10.9 (3.8)
Segmentation	17.1 (4.1)	7.1 (3.5)	7.0 (2.3)	12.0 (4.4)
Sonar	17.9 (3.3)	16.7 (2.2)	15.3 (4.9)	8.7 (3.1)
Thyroid	11.5 (7.0)	14.1 (0.9)	12.8 (0.9)	11.6 (1.1)
Vehicle	7.3 (4.7)	12.6 (4.3)	11.2 (3.2)	8.4 (4.1)
Vertebral	5.9 (4.2)	12.5 (1.7)	12.1 (3.1)	12.5 (1.7)
WBC	4.4 (0.5)	12.4 (1.5)	8.2 (2.6)	12.4 (1.5)
WDVG	9.2 (6.0)	11.3 (2.6)	11.6 (3.0)	11.3 (2.6)
Weaning	10.6 (1.6)	11.9 (1.6)	18.1 (4.8)	11.9 (1.6)
Wine	11.2 (0.9)	12.8 (1.9)	14.8 (5.6)	12.8 (1.9)
Average	10.3 (3)	11.9 (2.8)	10.7 (2.8)	11.9 (2.9)

population P, named S_i and S_j. Each individual is a data subset for which we have the corresponding difficulty computed using the selected complexity measures. Thus, the crossing operator combines the two selected parents' genes to generate a child. In our method, the crossing generates a new subset of data exchanging part of the instances between the parents S_i and S_j. Algorithm 3 details the crossover operation. Line 2 extracts all the problem classes. Next, the $CrossValid$ variable is set as $True$, meaning that we previously assumed that the crossing is valid, i.e., the generated subset has instances of all problem classes. In lines 5 and 6, two arbitrary subsets of data S_i and S_j are chosen from the population P, and the resulting (child) subset S_{new} is initialized as an empty structure (line 7). Then, the crossover points `beginning` and `finish` are defined randomly (lines 8–9). They are used to determine the interval over which instances are exchanged between S_i and S_j (lines 10 to 17). Finally, the algorithm checks if S_{new} has instances of all problem classes (lines 18–31).

Fig. 2 illustrates the crossover process between two data subsets (S_i and S_j). The beginning and finish points are chosen at random and determine where the exchange of instances between the data subsets will take place. As we can see, the new data subset (S_{new}) is created using instances from $S_i = x_4, x_{15}..., x_n$ and $S_j = x_5, x_{12}, x_{14}$, as we emphasize using the orange rectangle.

4.2.4. Mutation

During mutation in a genetic algorithm, a gene experiences a change. The individual who suffers the modification has a new characteristic that can contribute to the evolution of the population. In our method, a subset of data is modified in an instance. To this end, two subsets are chosen within the current population, one subset "donor" and one "receiver". The donor subset replaces an instance of the receiving subset. The donated instance and the modified one are chosen randomly within each subset, but both must have the same label.

Fig. 3 represents the mutation process, where line 1 shows the subset S_i that was chosen randomly from the population to be the receptor. In the same way, the subset S_j was chosen to be the donor.

Algorithm 1: Genetic Algorithm

Input : ψ as maximum number of generations,
κ as the population size,
θ as the number of new individuals after crossover and mutation,
T_r as training set,
V as validation set,
I as base inducer,
Op as method variant
Output : ϱ ensemble of classifiers.

1 **Function** GA(ψ, κ, θ, T_r, V, I, Op):
2 $t \leftarrow 0$
3 $P(t) \leftarrow$ initialize(T_r, κ)
4 $F(t), E(t) \leftarrow$ fitness_evaluate($P(t), \kappa, I$)
5 **while** $t \neq \psi$ **do**
6 $\gamma \leftarrow 0$
7 $P'(t) \leftarrow \varnothing$
8 **while** $\gamma \neq \theta$ **do**
9 $operator \leftarrow$ randomly_select(crossover, mutation)
10 $P'(t) \leftarrow P'(t) \cup$ operator($P(t)$)
11 γ ++
12 **end**
13 $F(t), E(t) \leftarrow$ fitness_evaluate($P'(t), \kappa, I, V$)
14 $P(t+1), F(t+1), E(t+1) \leftarrow$ NSGAII($P'(t), P(t), F(t), E(t), \kappa$) // see [10]
15 **if** $Op = PGDCS\_A$ **then**
16 $\upsilon \leftarrow accuracy(E(t+1), V)$ // using Majority Voting Rule
17 **end**
18 **else if** Op=$PGDCS\_D$ **then**
19 $\upsilon \leftarrow G_{disp}(F(t+1))$ // using Eq. (13)
20 **end**
21 $\varrho \leftarrow E(t+1)$
22 **if** $t = 0$ **then**
23 $temp \leftarrow \upsilon$
24 **else if** $temp < \upsilon$ **then**
// keeping the best generation
25 $temp \leftarrow \upsilon$
26 **end**
27 $P(t) \leftarrow P(t+1)$
28 $t \leftarrow t+1$
29 **end**
30 **return** ϱ

The instance x_2 of the subset S_3 replaces the x_7 example of the S_i subset, thus modifying the subset S_i and keeping the subset S_j (line 3) unchanged.

4.3. A note on complexity analysis

Assuming a dataset with n instances, f features, and y classes, the first part of proposed method is run r times. In each run, a number of bags N is created with random sampling of the original dataset. Next, the complexity metrics are computed along with their standard deviation, and their aggregated complexity is $O(f^3y^2)$ (proof in [36]). Consequently, the complexity of the first step is $O(rn(f^3y^2 + N))$. The second step regards the genetic algorithm optimization process, in which NSGA-II [10] was applied. This process is repeated r' times, such that each run encompasses g generations. For each generation g, NSGA-II has a time complexity of $O(\mu \log \mu^{F_n-2})$, where F_n is the number of objectives being optimized. Nonetheless, the computation of the objectives depend on complexity metrics, which as depicted above, have an $O(f^3y^2)$ cost. Therefore, the time complexity of the

Table 7
Average Oracle of each pool of classifiers over 20 iterations.

Dataset	Perceptron						Decision Tree					
	PGDCS_A	PGDCS_D	Bagg.	AdaB.	RF	DSOC	PGDCS_A	PGDCS_D	Bagg.	AdaB.	RF	DSOC
Australian	99	99	99	100	100	99	99	99	99	81	100	99
Banana	99	100	99	99	99	99	99	99	99	95	99	99
Blood	99	99	99	100	97	97	97	98	98	100	97	97
CTG	99	99	99	98	100	99	99	100	99	95	100	99
Diabetes	99	99	100	100	100	99	100	100	99	90	100	99
Faults	99	98	99	99	99	99	99	99	99	98	99	99
German	99	99	99	100	100	99	100	99	99	93	100	99
Haberman	100	100	100	100	99	98	99	100	99	92	99	98
Heart	99	99	99	100	100	99	100	100	99	84	100	99
ILPD	99	99	99	100	99	99	100	100	99	78	99	99
Ionosphere	100	99	99	99	100	99	99	99	99	87	100	99
Laryngeal1	100	100	99	100	99	99	100	100	99	87	99	99
Laryngeal3	99	99	99	100	100	99	99	99	99	74	100	99
Lithuanian	99	100	100	100	99	99	99	99	99	95	99	99
Liver	100	100	99	100	100	100	100	100	100	80	100	100
Mammo	100	99	99	100	97	97	98	98	97	100	97	97
Monk	100	100	100	100	100	100	100	100	100	100	100	100
P2	100	100	100	100	100	99	100	100	100	91	100	99
Phoneme	100	100	100	100	99	99	99	99	99	84	99	99
Segmentation	99	99	99	99	100	99	100	99	99	95	100	99
Sonar	100	99	100	100	100	100	100	100	100	89	100	100
Thyroid	99	99	100	100	99	99	99	99	99	95	99	99
Vehicle	99	100	99	99	100	100	100	100	100	88	100	100
Vertebral	99	99	99	100	100	99	99	100	99	83	100	99
WBC	100	99	99	99	99	99	100	99	99	92	99	99
WDVG	99	99	99	100	100	99	100	100	100	84	100	99
Weaning	99	99	100	100	100	99	100	100	99	97	100	99
Wine	100	100	100	97	100	100	100	100	100	90	100	100
Average	**99**	**99**	**99**	**99**	**99**	**99**	**99**	**99**	**99**	**90**	**99**	**99**

Table 8
Experiments on Time Consuming. Number of times the proposed method is slower than Bagging (Bag), AdaBoost (Ada) and Random Forest (RF). For each problem: Name (#samples, #classes, #features).

Classification Problem	PGDCS/Bag	PGDCS/Ada	PGDCS/RF
Wine (178, 3, 13)	9	20	21
WDVG (5,000, 3, 30)	20	98	112
Lithuanian (2000, 2, 2)	51	117	120
Faults (1941, 7, 27)	15	66	73

Algorithm 2: Fitness_evaluate

Input : P as the population,
κ as the population size,
I as base inducer
V as validation set
Output: F as a structure with three fitness values for each individual,
E as the ensemble of classifiers

Function `fitness_evaluate(`P, κ, I, V`)`:
  for *each* S_i ***in*** P **do**
    Compute metric cm_1 as $\varphi_{cm_1}[S_i]$
    Compute metric cm_2 as $\varphi_{cm_2}[S_i]$
    $C_i \leftarrow$ Training the base inducer I_i on S_i
  end
  $E \leftarrow \emptyset$
  for i ***in*** $[1..\kappa]$ **do**
    Compute $F[i][0]$ as $\Phi_{cm_1}[S_i]$ (Eq. (14))
    Compute $F[i][1]$ as $\Phi_{cm_2}[S_i]$ (Eq. (14))
    Compute $F[i][2]$ as $DDV(C_i)$ using V (Eq. (15))
    $E \leftarrow E \cup C_i$
  end
return F, E

second step of the proposed method is of $O(g \times r' \times (\mu \log \mu^{F_n-2} f^3 y^2))$. As a result, the overall time complexity of the proposed method is $O(rn(f^3y^2 + N) + g \times r' \times (\mu \log \mu^{F_n-2} f^3 y^2))$.

5. Experimental results and analysis

A robust experimental protocol is used to evaluate the two variants of the proposed method, PGDCS_A (*Maximum Accuracy*), and PGDCS_D (*Maximum Dispersion*). We used 28 datasets available in the UCI [27], KEEL [31], PRTools [28], STATLOG [29], LKC [30] and ELENA [32] repositories and P2 Dataset [33]. Table 2 presents the datasets with their main characteristics and respective repositories.

We compare the proposed method with traditional methods of pool generation, Bagging [4], AdaBoost [5], Random Forest [7] and the first part of the DSOC framework that regards pool generation [16]. Besides, we evaluate the impact of using the proposed method as pool generator for well-known DCS methods like LCA, OLA, and Rank [37], and DES methods like KNORA-E, KNORA-U [38], and META-DES [39,40]. The DCS and DES methods are publicly available as part of the `deslib` library [25]. Table 1 presents the parameters of each method used in our experiments.

As base inducer for homogeneous pool generation, we considered Perceptron and Decision tree. Perceptron was selected given the promising results when used in dynamic selection methods in [41], while the decision tree is an unstable inducer that results in higher pool diversity.

5.1. Selection of data complexity metrics

We first use the Lithuanian Problem (Fig. 4) to explain how we found the most promising pair of complexity measures for a given classification problem. The motivation for this classification problem is its simplicity. Next, we run the same procedure for the entire set of classification problems used in our experiments.

The Lithuanian dataset is divided into training, validation, and testing, with a proportion of 50%, 25%, 25%, respectively. With the

Table A.9
Results of combining classifiers by majority vote using Perceptron as base classifier (accuracy and Std. Dev.).

Data	MVR - Perceptron					
	PGDCS_A	PGDCS_D	Bagging	AdaBoost	R.F.	DSOC
Australian	87.8 (2.2)	**88.1 (2.3)**	86.9 (2.1)	86.9 (2.2)	87.7 (2.0)	85.8(2.0)
Banana	84.7 (1.3)	83.5 (2.0)	84.6 (1.4)	83.9 (1.5)	**97.2 (0.8)**	83.5(1.9)
Blood	77.1 (3.3)	76.7 (1.9)	**77.6 (2.0)**	77.5 (1.3)	74.9 (2.8)	74.1(3.9)
CTG	88.9 (1.7)	84.7 (17.7)	81.1 (24.3)	81.1 (23.4)	**93.5 (1.0)**	80.8(24.2)
Diabetes	**76.9 (3.0)**	76.4 (2.7)	76.8 (2.8)	76.6 (2.7)	75.7 (3.3)	74.7(3.1)
Faults	70.7 (2.1)	70.5 (1.4)	69.4 (1.7)	69.5 (1.3)	**75.9 (1.6)**	67.6(2.2)
German	**77.0 (2.4)**	76.5 (3.0)	75.9 (2.3)	75.8 (2.6)	74.8 (2.5)	74.2(3.0)
Haberman	**76.8 (3.3)**	**76.8 (2.7)**	75.0 (2.3)	74.7 (2.0)	69.7 (3.7)	74.3(2.8)
Heart	**85.9 (3.9)**	85.4 (3.6)	82.5 (3.7)	81.5 (4.6)	80.0 (4.8)	82.3(2.7)
ILPD	70.6 (3.7)	**72.7 (3.3)**	71.4 (2.7)	71.2 (3.2)	70.4 (2.8)	69.4(3.1)
Ionosphere	89.9 (2.7)	91.4 (3.8)	89.2 (2.9)	87.0 (4.9)	**93.4 (2.9)**	88.5(3.4)
Laryngeal1	**86.0 (5.1)**	84.2 (3.7)	83.5 (3.7)	81.9 (4.8)	83.2 (3.9)	83.5(5.0)
Laryngeal3	**76.3 (3.2)**	74.8 (2.7)	72.7 (2.3)	68.7 (7.7)	72.3 (2.8)	71.9(3.4)
Lithuanian	81.6 (2.1)	82.9 (1.7)	82.8 (1.7)	82.4 (1.8)	**97.0 (0.6)**	82.0(1.8)
Liver	**70.5 (3.3)**	68.4 (3.8)	68.7 (4.2)	68.0 (3.9)	68.4 (4.0)	64.9(5.8)
Mammo	**84.0 (2.3)**	83.8 (2.5)	82.9 (2.1)	81.9 (2.4)	78.7 (2.0)	82.1(2.6)
Monk	85.7 (2.3)	83.4 (2.6)	78.9 (3.2)	78.0 (3.5)	**98.8 (1.6)**	80.0(3.7)
Phoneme	54.6 (4.1)	76.1 (1.4)	75.6 (1.1)	75.9 (0.7)	**89.1 (0.8)**	74.6(2.6)
P2	77.6 (1.1)	56.7 (4.4)	56.5 (4.6)	51.9 (3.9)	**93.7 (1.0)**	53.3(3.4)
Segmentation	90.8 (1.0)	91.0 (1.4)	91.0 (1.2)	91.5 (1.1)	**97.0 (1.0)**	90.4(1.0)
Sonar	**84.4 (6.2)**	81.7 (6.5)	77.7 (3.4)	72.3 (7.2)	78.1 (3.9)	75.9(3.5)
Thyroid	97.1 (1.0)	**97.3 (0.9)**	96.6 (1.2)	96.8 (1.3)	96.0 (0.8)	96.2(1.6)
Vehicle	75.5 (2.6)	**77.5 (2.4)**	75.2 (2.0)	75.4 (2.0)	74.5 (2.1)	73.6(2.4)
Vertebral	**89.1 (3.8)**	87.9 (3.6)	87.1 (3.4)	86.5 (3.3)	85.7 (3.2)	86.1(3.4)
WBC	98.0 (0.8)	**98.2 (1.0)**	97.1 (1.1)	95.1 (3.9)	95.5 (1.5)	96.9(1.0)
WDVG	**86.8 (1.0)**	**86.8 (0.9)**	86.5 (0.8)	86.7 (0.9)	85.0 (0.7)	85.6(0.9)
Weaning	85.3 (3.6)	84.9 (3.6)	82.5 (3.8)	81.8 (4.0)	**89.5 (2.8)**	81.7(3.8)
Wine	**98.4 (1.9)**	98.0 (2.3)	97.5 (2.3)	97.2 (2.6)	97.3 (2.8)	97.3(1.7)
Average	82.4	82.0	80.8	79.9	**84.8**	79.7

training set and considering $n = 100$, and $u = 0.4$, we randomly sampled 100 data subsets considering each a maximum of 50% of the original training size. For each data subset S_i, the two families of complexity measures are computed. By defining $T = 10$, this procedure is repeated ten times, producing 1000 data subsets.

Table 3 shows the average dispersion of the Overlapping and Neighborhood measures at each iteration for the Lithuanian dataset. The results show that the metrics with the most votes are F3 and T1, with 4 and 3 votes, respectively.

Table 4 summarizes the results, showing the complexity measures and corresponding votes computed by the first step of the proposed method. As we can see, selecting the best metric is problem-dependent since we did not observe any tendency.

5.2. Pool generation using MOGA

After choosing the most promising metrics for each classification problem, our method starts generating the pool of classifiers. We used the NSGA-II [10] multi-objective genetic algorithm (implemented in `DEAP 1.0` [26]) and the Double Fault diversity measure in that second step.

Table 5 shows the MOGA configuration used in our method. As already mentioned, we have evaluated Perceptron and Decision Tree classifiers as base inducers. Finally, we have performed 20 replications, providing the average accuracy and corresponding standard deviation as final results.

The first generation of our MOGA consists of data subsets with just 50% of the training dataset size. Table 6 presents on average the generation number that provided the maximum accuracy and maximum diversity within the 20 iterations for each classification problem, considering both inducers, Perceptron, and Decision Tree.

Using the PGDCS_D variant, the Banana dataset reached its maximum dispersion (using the Perceptron classifier) after eight generations in all replications. Similarly, when using PGDCS_A, the P2 classification problem gets the utmost accuracy in two generations. We can see that 20 generations seem to be enough since, in most cases, the average is less than 15.

With the Lithuanian dataset again, we have plotted in Fig. 5 the distribution of the data subsets over the space formed by the two selected complexity measures (F3 and T1) and the diversity measure (Double Fault). Fig. 5(a) presents the distribution of the first MOGA generation. In contrast, Fig. 5(b) shows the distribution of the best generation (iteration 15, generation 18) using PGDCS_D variant and Decision Tree as base inducer. Each point in this plot is a data subset generated by the proposed method, Φ_{F3} is the average pairwise distance of the overlapping complexity measure $F3$, Φ_{T1} is the average pairwise distance of the neighborhood complexity measure $T1$. At the same time, DDV represents the diversity (Double Fault) computed for the classifier trained on each subset in a pairwise manner. As we can see, the individuals (subsets of data) of the best generation better cover the problem complexity space even considering an additional objective function devoted to diversity in the decision space.

Similar behavior was observed for the other classification problems. This can contribute to validate our hypothesis that the proposed sampling oriented by data complexity measures generates data subsets that better cover the data complexity space. However, we cannot say whether such behavior may contribute to the pool's performance. Such an investigation is the subject of the next section.

5.2.1. Combining the pool members using majority vote rule (MVR)

To answer our first research question (RQ1), we need to show whether the proposed method oriented by a two-level diversity strategy is able or not to compete with traditional pool generation methods. To this end, we compared the results obtained by the two variants of the PGDCS method with some traditional methods (Bagging, AdaBoost, RF) and DSOC that is also oriented by complexity measures. The Majority Vote Rule (MVR) was used to combine the pool members.

In Appendix, Tables A.9 and A.10 present the results of the two PGDCS variants (PGDCS_A, PGDCS_D) for all datasets using Perceptron and Decision Trees as base classifiers, respectively. Such results represent the average and standard deviation of 20 iterations. To facilitate

Algorithm 3: Crossover

```
Input  : P as the population
Output: S_new as a new individual (data subset)
Function Crossover(P):
    Y ← getClass(S_i) // returns the classes present in S_i
    ValidCross ← False
    while ValidCross ≠ True do
        S_i ← select randomly an individual from P
        S_j ← select randomly an individual from P \ {S_i}
        S_new ← ∅
        beginning← select randomly an index in the range
          [0, |S_i| − 1]
        finish← select randomly an index in the range
          [beginning, |S_i| − 1]
        for k to max(|S_i|, |S_j|) do
            if k ≤ beginning or k ≥ finish then
                S_new ← S_i(x_k, y_k)
            end
            else
                S_new ← S_j(x_k, y_k)
            end
        end
        foreach class in Y do
            count_samples ← 0
            foreach sample in S_new do
                if label(sample) = class then
                    count_samples++
                end
                if count_samples >= 2 then
                    ValidCross ← True
                else
                    ValidCross ← False
                end
            end
        end
    end
return S_new
```

the visualization and analysis, Fig. 6 shows the results in a win/tie/loss perspective. The win–tie–loss analysis allow us to compute a critical level (Nc). It represents the number of wins plus half the number of ties and must be greater than Nc (Eq. (17)), where $N_{exp} = 28$ (number of experiments), $Z_c = 2.57$ for the significant level 0.01, then $Nc = 20.8$.

$$Nc = \frac{N_{exp}}{2} + Z_c \frac{\sqrt{N_{exp}}}{2} \tag{17}$$

As we can see, considering the 28 classification problems and the Perceptron as base inducer, when compared to Bagging, AdaBoost, and RF, the PGDCS_A achieved 90 wins (80.3%), two ties, and 20 losses over a total of 112 comparisons. The results are still more promising when we used Decision Tree as base classifier. In this case, the PGDCS_A achieved 109 wins (97.3%), two ties, and one loss. As one may also see in Fig. 6, the performance of the PGDCS_D was a little inferior. The PGDCS_D with Perceptron achieved 91 wins (81.2%), three ties, and 18 losses over a total of 112 comparisons. Similarly, the results are more promising when we used Decision Tree as a base classifier. In this case, the PGDCS_D achieved 108 wins (96.4%) and four losses.

A statistical analysis based on the Friedman and Nemenyi tests [42] with a p-value = 0.01 confirmed a significant difference between the compared methods. Fig. 7 presents the rankings produced by means of the Nemenyi post-hoc test.

In general, we observed that the proposed method is a promising strategy to generate a pool of complementary classifiers. The idea of training the classifiers on data subsets representing sub-problems with different levels of difficulty has contributed to improving the pool accuracy. Besides, a more unstable base inducer significantly improved the results of both variants of the proposed method.

5.2.2. Results of dynamic selection methods

As already mentioned, we expect that the pool generated can contribute to improving the performance of MCS based on dynamic selection methods. With this in mind, we have evaluated our method in the context of dynamic classifier selection (DCS) and dynamic ensemble selection (DES). Table 7 presents the performance (accuracy) of the oracle considering each pool generator evaluated in our experiments. It is essential to mention that the concept of oracle in dynamic selection methods is the upper limit in performance that a given pool can reach. According to Kuncheva [3], Oracle is an abstract concept of fusion. In this model, if at least one of the classifiers produces the correct class label, the pool can also make it. As we can see, except for Adaboost, most of the time, the oracle is at least 99%.

Given the generated pools, to answer our second research question (RQ2), we first applied three dynamic classifier selection methods, considering an experimental protocol with 20 replications. The average results considering Perceptron and Decision Tree as base inducers are presented in Appendix, Tables A.11 and A.12, respectively.

We have summarized the results related to DCS methods in Figs. 8 and 9. These plots show the performance of the two variants of the proposed method in terms of win/tie/loss when using Perceptron and Decision tree as base inducers. As expected, the use of a Decision tree is a better option. Besides, the PGDCS_A has again shown a better performance. This means that diversity alone is not enough to contribute to the performance of DCS methods. In summary, we have done 336 experiments considering DCS methods for each proposed variant. In general, considering the best inducer (Decision Tree), the proposed PGDCS_A (Fig. 8) achieved 315 wins (93.7%), three ties, and 18 losses, while the best results with PGDCS_D (Fig. 9) was 314 wins (93.4%), three ties, and 19 losses. In all experiments, the DCS methods were executed using the default parameters as shown in Table 1 (pool size = 100, and neighborhood size = 7), but using a different pool generated by Bagging, AdaBoost, RF, PGDCS_A, or PGDCS_D.

A statistical analysis based on the Friedman and Nemenyi tests [42] with a p-value = 0.01 confirmed a significant difference between the compared methods. Figs. 10 and 11 present the rankings produced by means of the Nemenyi post-hoc test.

The most significant impact was observed in the approaches that considers the overall accuracy of the classifiers to perform the dynamic selection. The OLA and Rank methods for which both variants (PGDCS_A and PGDCS_D) had 109 (97.3%) wins, one tie, and two losses.

In another set of experiments we have evaluated the impact of the proposed method in MCS based on DES. The average results considering Perceptron and Decision Tree as base inducers were provided in Appendix, Tables A.13 and A.14, respectively.

We have summarized the results related to DES methods in Figs. 12 and 13. These plots show the performance of the two variants of the proposed method in terms of win/tie/loss when using Perceptron and Decision tree as base inducers. The DES methods were executed using the same parameters (poll size = 100, and neighborhood size = 7), but different pools generated by Bagging, AdaBoost, RF, PGDCS_A, or PGDCS_D. Finally, PGDCS_A using a Decision tree is again the best option.

A statistical analysis based on the Friedman and Nemenyi tests [42] with a p-value = 0.01 confirmed a significant difference between the compared methods. Figs. 14 and 15 present the rankings produced by means of the Nemenyi post-hoc test.

We can see a significant impact of the variant PGDCS_A on all DES methods evaluated. The most important was the one observed on KNORA-U, where 109 (97.3%) wins, two ties, and one loss over 112 experiments were observed.

The proposed method using the Decision Tree as the base classifier generated pools that were superior to all the competitors. The best results were achieved with the PGDCS_A approach.

Table A.10
Results for combining classifiers by majority vote using Decision tree as base classifier (accuracy and Std. Dev.).

Data	MVR - Decision Tree					
	PGDCS_A	PGDCS_D	Bagging	AdaBoost	R.F.	DSOC
Australian	**93.0(2.3)**	91.3(2.8)	87.6(1.6)	82.3(2.6)	87.7(2.0)	86.6(1.9)
Banana	**97.8(0.7)**	97.6(0.7)	97.0(0.8)	95.8(0.8)	97.2(0.8)	96.8(1.0)
Blood	**83.6(2.4)**	81.7(2.9)	77.4(3.2)	74.4(2.5)	74.9(2.7)	75.4(3.8)
CTG	**95.8(1.0)**	95.5(1.0)	93.1(1.1)	92.0(1.5)	93.5(1.0)	92.8(0.9)
Diabetes	**85.2(2.8)**	81.8(4.6)	75.8(2.8)	69.0(3.8)	75.7(3.3)	75.3(3.5)
Faults	83.6(5.0)	**84.6(2.9)**	75.4(1.7)	69.4(2.7)	75.9(1.6)	73.8(2.2)
German	**85.9(5.1)**	82.5(5.2)	74.9(2.7)	68.3(3.1)	74.8(2.5)	73.5(3.2)
Haberman	**80.5(6.1)**	79.5(5.2)	70.4(4.1)	71.4(4.5)	69.7(3.7)	71.6(4.2)
Heart	**90.1(4.1)**	88.4(5.2)	80.4(4.5)	73.5(5.1)	80.0(4.8)	79.3(6.0)
ILPD	81.6(5.6)	**81.7(4.8)**	69.9(3.2)	65.8(3.9)	70.4(2.8)	69.6(4.4)
Ionosphere	**96.2(1.5)**	95.9(1.9)	92.4(2.7)	87.4(4.5)	93.4(2.9)	91.8(3.0)
Laryngeal1	**91.5(4.6)**	88.4(5.1)	83.1(4.7)	77.8(4.7)	83.2(3.9)	82.0(5.1)
Laryngeal3	81.0(5.5)	**81.1(4.9)**	73.3(3.1)	63.2(4.6)	72.3(2.8)	71.6(2.9)
Lithuanian	**98.0(0.8)**	97.8(0.5)	97.0(0.6)	95.6(0.8)	97.0(0.6)	96.7(0.6)
Liver	**85.1(3.2)**	79.7(7.5)	69.4(3.7)	60.2(4.1)	68.4(4.0)	66.9(3.4)
Mammo	**85.4(4.1)**	84.4(3.9)	81.5(1.8)	55.9(4.5)	78.7(2.0)	79.7(1.8)
Monk	**100.0(0.0)**	99.9(0.4)	**100.0(0.2)**	**100.0(0.0)**	98.8(1.6)	99.6(1.3)
Phoneme	**96.8(1.2)**	95.5(1.2)	93.1(1.0)	91.7(1.7)	93.7(1.0)	87.1(0.9)
P2	91.9(2.6)	91.7(2.2)	88.0(0.9)	84.8(1.1)	89.1(0.8)	**92.4(1.3)**
Segmentation	**98.1(0.9)**	97.9(0.8)	96.3(0.9)	95.3(1.1)	97.0(1.0)	95.8(0.9)
Sonar	84.7(5.5)	**85.5(6.6)**	74.3(5.4)	69.4(6.8)	78.1(3.9)	72.6(6.1)
Thyroid	97.7(1.2)	**97.8(0.9)**	95.9(1.0)	95.1(0.9)	96.0(0.8)	95.7(1.0)
Vehicle	**86.1(3.4)**	85.2(5.1)	72.9(1.8)	68.3(2.5)	74.5(2.1)	72.3(3.0)
Vertebral	92.1(4.1)	**92.5(4.0)**	85.7(3.3)	83.4(3.3)	85.7(3.2)	84.1(3.7)
WBC	**97.0(1.7)**	96.6(1.9)	94.7(1.8)	92.3(1.9)	95.5(1.5)	94.4(2.0)
WDVG	**90.5(2.5)**	89.5(3.0)	84.3(0.9)	74.6(1.1)	85.0(0.7)	83.1(0.9)
Weaning	**93.7(3.4)**	90.1(3.0)	85.7(4.0)	77.1(5.5)	89.5(2.8)	83.6(3.9)
Wine	**99.3(1.0)**	96.4(2.2)	95.2(4.4)	89.4(5.9)	97.3(2.8)	94.2(4.2)
Average	**90.8**	89.6	84.4	79.4	84.7	83.5

5.3. Time consumption evaluation

We have performed experiments concerning the time-consuming. When considering the 28 classification problems of our experimental protocol, the PGDCS method consumes, on average, 30, 75, and 80 times more processing time than Bagging, Adaboost, and Random Forest, respectively. Table 8 shows a comparison considering four of our classification problems. They were selected considering the number of samples and classes. Wine has a small number of samples (178), while WDVG is the biggest one with 5,000 samples. The Lithuanian is a binary problem (2 classes), while the Faults problem has seven classes.

As we can see, the time consuming depends on the classification problem and the complexity measures selected for the second-stage of the proposed method. This is why datasets showing a similar number of samples have an important difference in time consumption. Finally, it is important to notice that this time is related to the pool generation process (first and second steps), performed only during the construction of the classification method.

6. Conclusion

In this paper, we proposed a novel classifier pool generation method and its variants: maximum dispersion (PGDCS_D) and maximum accuracy (PGDCS_A). In both cases, a sampling process organizes data subsets for training the classifiers using a two-level strategy to promote diversity. Such a sampling strategy considers diversity in both data complexity and classifiers' decision spaces.

A robust experimental protocol based on 28 classification problems showed that the proposed method overcomes traditional methods like Bagging, Adaptive Boosting, and Random Forest, especially when dynamic selection of classifiers and ensembles is applied. The best results of the proposed method were obtained with the variant PGDCS_A using Decision Tree as base inducer. However, PGDCS_D also provided competitive results when used to provide the pool for MCS with dynamic selection (DS). In summary, we have done 336 experiments. In general, the best PGDCS_A achieved 327 wins (97.3%), seven ties, and two losses, while the best PGDCS_D achieved 324 wins (96.4%), seven ties, and five losses.

The PGDCS proved to be an interesting ensemble learning method mainly when employed with dynamic classifier selection. However, the method has unbalanced datasets as a limitation, as some complexity measures require at least two examples of different labels for each bag. As PGDCS uses random sampling in both steps, such a criterion may not be satisfied.

As future works, we propose integrating the first and second steps of the proposed method in a singular optimization process in which we could consider different complexity metrics over the generations. In such a direction, we plan to investigate a cascading evolutionary algorithm, so at the end of each generation, the complexity measures are verified to check whether they still have the same influence on the bags' evolution.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

I shared the link at the paper text

Acknowledgments

We thank STICAmSud (project 19-STIC-04) and CNPq Brazil (project 306684/2018-7) that co-financed this work.

Table A.11
Comparison of DCS methods using different pools of classifiers with Perceptron as base inducer (accuracy and Std. Dev.).

Data	LCA						OLA						Rank					
	PGDCS_A	PGDCS_D	Bagging	AdaB.	R.F.	DSOC	PGDCS_A	PGDCS_D	Bagging	AdaB.	R.F.	DSOC	PGDCS_A	PGDCS_D	Bagging	AdaB.	R.F.	DSOC
01	**84.6(3.4)**	84.0(2.8)	82.9(2.7)	83.1(2.7)	82.3(3.0)	81.3(2.6)	83.0(2.2)	**83.3(2.5)**	82.9(1.9)	82.8(2.0)	79.0(1.9)	**83.3(3.2)**	81.1(2.9)	81.5(2.8)	81.4(2.7)	81.3(2.6)	79.0(2.1)	**81.9(3.8)**
02	95.7(1.0)	95.7(1.0)	95.7(0.9)	95.7(1.0)	**97.1(0.7)**	95.6(1.0)	**97.4(0.7)**	**97.4(0.6)**	**97.4(0.6)**	**97.4(0.7)**	96.5(0.6)	97.0(0.7)	**96.9(0.8)**	**96.9(0.7)**	**96.9(0.8)**	**96.9(0.8)**	96.5(0.7)	96.5(0.9)
03	68.9(3.9)	72.9(5.3)	70.4(5.3)	72.6(3.6)	**74.6(2.4)**	69.7(5.3)	75.2(3.3)	**75.7(3.0)**	74.8(2.5)	74.5(2.7)	72.1(3.8)	75.1(3.7)	70.0(4.2)	70.3(3.6)	69.8(3.7)	69.5(2.8)	**71.3(3.5)**	70.2(5.1)
04	86.8(1.8)	86.7(1.7)	86.2(1.9)	86.7(2.2)	**89.4(1.2)**	86.5(2.1)	89.3(1.2)	89.2(1.3)	89.0(1.4)	88.8(1.7)	**90.2(1.5)**	88.8(1.4)	89.3(1.4)	89.3(1.3)	88.9(1.6)	88.6(2.7)	**90.2(1.5)**	88.7(2.0)
05	69.9(3.3)	69.8(3.2)	**70.7(3.1)**	69.8(3.1)	68.1(2.6)	69.1(3.0)	**73.3(3.3)**	73.1(3.1)	72.4(2.6)	72.0(2.9)	68.6(2.3)	71.9(2.2)	**71.8(2.6)**	71.5(3.3)	71.5(2.6)	70.8(3.0)	68.5(3.0)	70.5(2.9)
06	64.5(1.9)	**65.0(2.4)**	62.8(2.4)	63.8(1.6)	64.8(1.9)	63.0(2.4)	67.4(1.5)	**68.7(1.5)**	67.7(1.9)	67.9(1.4)	66.4(1.8)	67.6(1.8)	67.2(1.8)	**68.1(1.6)**	67.4(2.1)	67.7(1.4)	66.2(2.4)	67.4(2.1)
07	**70.1(2.6)**	69.1(3.1)	69.1(2.2)	68.5(3.1)	67.3(2.6)	68.2(2.5)	70.4(2.1)	**71.3(2.6)**	71.2(2.3)	69.8(1.9)	67.0(3.0)	71.0(2.0)	69.4(1.9)	70.1(2.5)	**71.0(2.7)**	69.1(2.2)	66.6(3.1)	70.1(2.6)
08	62.0(7.4)	**72.8(3.7)**	69.0(7.8)	67.2(10.3)	70.7(4.1)	66.7(8.4)	69.5(4.4)	68.4(4.9)	69.7(4.5)	69.0(5.9)	66.2(5.2)	**69.8(5.2)**	66.0(5.1)	64.7(6.2)	65.7(5.2)	65.4(5.6)	65.8(5.3)	**66.5(5.9)**
09	80.1(5.5)	**80.6(5.2)**	75.7(7.1)	79.6(6.0)	72.5(5.9)	79.2(5.3)	**78.5(4.6)**	78.4(5.7)	77.5(4.5)	76.0(4.0)	71.9(5.2)	78.4(5.4)	**77.5(4.1)**	76.9(5.0)	77.1(4.5)	74.8(4.5)	72.0(5.0)	77.2(6.2)
10	**69.7(2.1)**	66.9(3.2)	66.8(4.8)	66.2(4.5)	67.9(4.2)	64.6(4.4)	**69.3(3.0)**	69.1(3.3)	69.0(3.5)	68.8(3.1)	67.3(3.7)	69.1(2.4)	67.7(3.8)	67.8(3.4)	68.0(2.8)	**68.2(3.0)**	67.5(3.7)	67.1(3.7)
11	**87.1(2.5)**	86.1(3.6)	85.4(3.9)	85.2(3.0)	84.7(4.0)	84.9(4.1)	**88.8(2.8)**	88.4(3.0)	85.9(2.7)	87.6(3.4)	87.1(3.5)	86.4(4.0)	**88.8(2.9)**	88.3(3.2)	86.0(2.8)	87.0(3.5)	87.2(3.5)	86.5(4.1)
12	**82.5(3.6)**	75.1(5.2)	77.6(6.8)	78.1(7.0)	79.8(4.6)	77.9(5.5)	81.3(3.8)	**81.4(4.7)**	80.6(3.7)	80.0(3.8)	77.3(4.9)	79.4(4.7)	81.3(3.9)	**81.4(3.8)**	78.6(3.5)	79.0(4.3)	76.3(4.6)	79.3(4.7)
13	67.6(4.4)	**70.5(4.9)**	68.1(4.4)	70.3(4.0)	67.8(4.7)	67.7(4.9)	67.3(4.2)	67.9(3.8)	68.0(4.8)	**69.2(4.1)**	63.7(4.9)	67.4(4.9)	64.3(5.2)	**66.6(4.3)**	64.1(5.8)	64.0(5.8)	63.4(5.1)	65.3(5.7)
14	93.4(1.2)	92.8(1.1)	92.7(1.0)	93.0(1.2)	**96.2(0.8)**	92.8(1.4)	96.4(0.6)	96.6(0.8)	**96.7(0.7)**	96.5(0.8)	95.8(0.9)	96.4(0.7)	95.7(0.7)	**95.9(0.8)**	**95.9(0.9)**	95.7(0.9)	95.7(0.9)	95.7(0.7)
15	57.9(5.5)	**58.0(4.6)**	55.3(5.3)	57.2(5.0)	55.0(5.6)	55.6(5.6)	66.6(5.0)	65.5(5.4)	**68.2(3.9)**	65.8(4.0)	63.1(4.6)	67.6(3.4)	66.2(4.3)	65.6(5.5)	**66.4(4.1)**	63.8(4.9)	61.7(4.4)	65.9(3.7)
16	74.1(5.4)	**80.4(2.8)**	79.7(3.6)	77.4(4.6)	79.5(2.3)	78.0(5.6)	**81.1(2.2)**	80.8(2.5)	80.8(2.6)	80.5(2.7)	78.4(2.9)	80.8(3.3)	77.1(2.7)	76.8(3.2)	77.4(2.4)	76.2(2.8)	**78.0(2.8)**	77.7(3.0)
17	72.8(2.9)	73.8(4.7)	72.8(4.3)	71.6(5.4)	**93.0(3.5)**	79.5(1.8)	87.6(3.2)	87.8(3.1)	85.7(3.6)	84.9(2.7)	**97.5(2.2)**	85.7(3.9)	87.8(3.4)	87.6(3.7)	85.5(3.8)	85.2(2.9)	**97.5(2.2)**	85.6(3.4)
18	78.8(1.6)	76.2(1.9)	75.7(2.4)	75.5(1.7)	**82.8(1.1)**	70.2(4.8)	**89.9(1.4)**	83.0(0.9)	82.8(0.6)	82.5(0.7)	85.0(0.9)	82.8(0.6)	**91.0(1.3)**	83.3(0.9)	83.0(0.8)	83.0(0.7)	85.0(0.8)	83.3(1.0)
19	76.5(2.4)	74.0(2.7)	73.4(3.0)	79.0(2.3)	**88.4(1.8)**	75.5(1.5)	82.4(0.8)	88.0(2.2)	88.1(1.7)	89.7(1.6)	**91.2(1.3)**	90.5(1.3)	82.8(0.8)	89.8(2.3)	90.1(1.7)	91.0(1.5)	91.2(1.3)	**91.8(1.2)**
20	88.5(1.4)	88.8(1.2)	88.7(1.2)	89.4(1.7)	**92.4(1.2)**	88.4(1.5)	92.6(1.2)	92.7(1.2)	92.8(1.4)	93.4(1.3)	**93.8(1.4)**	92.1(1.4)	92.8(1.3)	93.0(1.2)	93.0(1.1)	93.5(1.1)	**93.9(1.4)**	92.5(1.3)
21	73.2(6.9)	**73.3(7.4)**	66.9(6.8)	66.6(7.8)	63.7(7.1)	70.4(5.7)	77.7(6.5)	**78.6(6.9)**	75.1(6.2)	75.2(5.2)	70.6(7.0)	76.5(5.3)	77.5(7.2)	**79.6(6.1)**	74.8(5.5)	76.2(4.4)	70.8(7.0)	76.8(5.5)
22	94.4(1.4)	**95.5(1.6)**	95.0(1.9)	95.4(1.6)	94.6(1.2)	94.4(1.9)	96.1(1.3)	96.0(1.5)	95.7(1.7)	**96.2(1.6)**	94.5(1.4)	95.8(1.4)	95.6(1.6)	**95.7(1.8)**	95.1(1.7)	95.5(1.7)	94.4(1.5)	95.5(1.7)
23	68.1(2.3)	**69.1(3.0)**	67.1(2.5)	68.8(3.1)	64.3(3.6)	67.5(2.4)	74.0(2.0)	74.0(2.7)	73.1(2.0)	**74.5(3.1)**	67.5(2.0)	73.4(3.0)	**74.5(2.6)**	74.1(3.1)	73.0(2.0)	74.0(2.6)	67.4(1.8)	73.7(2.8)
24	81.5(4.2)	82.7(3.4)	81.7(4.6)	**83.3(4.2)**	78.2(4.6)	79.3(6.4)	**85.1(4.0)**	83.8(3.9)	83.8(4.3)	83.5(4.0)	81.1(4.0)	84.9(4.7)	**84.6(4.0)**	83.6(3.6)	84.3(5.0)	83.0(4.2)	81.3(4.2)	84.1(4.8)
25	**95.5(1.4)**	95.3(1.8)	95.0(1.4)	95.4(1.5)	91.3(2.3)	94.5(1.4)	96.3(1.1)	**96.6(1.3)**	96.0(1.3)	96.0(1.2)	92.5(2.2)	96.0(1.4)	96.2(1.1)	**96.7(1.3)**	95.9(1.3)	95.8(1.4)	92.4(2.2)	96.1(1.3)
26	**80.3(1.6)**	78.9(1.8)	79.2(2.1)	79.9(2.0)	74.1(1.2)	79.1(2.2)	**83.0(1.1)**	82.9(0.9)	82.4(1.2)	82.6(1.0)	73.0(1.2)	82.8(1.0)	**82.4(1.4)**	82.3(0.9)	82.0(0.9)	82.2(1.0)	72.9(1.2)	82.2(1.0)
27	**78.7(4.9)**	77.3(4.4)	75.4(5.0)	76.9(5.0)	74.7(7.5)	76.3(5.1)	**81.4(4.1)**	80.3(5.2)	79.7(4.5)	79.3(4.2)	77.8(3.4)	79.2(4.8)	**81.3(4.7)**	80.1(4.9)	79.9(4.7)	77.2(4.7)	78.0(3.0)	79.4(5.4)
28	**97.0(1.9)**	96.1(2.4)	94.2(4.3)	**97.0(2.7)**	90.6(5.2)	94.3(3.6)	**97.3(2.0)**	95.6(2.7)	95.0(2.2)	97.0(2.7)	89.9(4.8)	95.7(3.0)	**97.3(2.0)**	95.6(2.7)	95.0(2.2)	97.0(2.7)	89.9(4.8)	95.7(3.0)
Aver.	78.6	**78.8**	77.6	78.3	**78.8**	77.5	**82.1**	81.9	81.5	81.5	79.5	81.6	**81.2**	**81.2**	80.6	80.4	79.3	80.8

Table A.12
Comparison of DCS methods using different pools of classifiers with Decision Tree as base inducer (accuracy and Std. Dev.).

Data	LCA						OLA						Rank					
	PGDCS_A	PGDCS_D	Bagging	AdaB.	R.F.	DSOC	PGDCS_A	PGDCS_D	Bagging	AdaB.	R.F.	DSOC	PGDCS_A	PGDCS_D	Bagging	AdaB.	R.F.	DSOC
01	85.4(2.1)	**86.1(2.8)**	84.5(1.9)	82.3(2.6)	82.3(3.0)	83.6(2.6)	**85.2(2.7)**	84.6(3.4)	82.4(3.1)	82.3(2.6)	79.0(1.9)	82.1(2.4)	**85.2(3.0)**	84.7(3.3)	82.1(2.7)	82.3(2.6)	79.0(2.1)	81.7(2.4)
02	97.0(0.8)	**97.1(0.6)**	96.9(0.7)	95.8(0.8)	**97.1(0.7)**	96.7(0.8)	**96.9(0.6)**	96.7(0.7)	96.5(0.8)	95.8(0.8)	96.5(0.6)	96.3(1.1)	**96.9(0.6)**	96.6(0.7)	96.4(0.8)	95.8(0.8)	96.5(0.7)	96.1(1.1)
03	**77.4(2.8)**	74.4(3.7)	74.5(2.4)	73.1(3.8)	74.6(2.4)	73.6(2.7)	**76.0(3.0)**	73.8(3.6)	72.6(2.8)	71.8(3.2)	72.1(3.8)	72.9(3.3)	**75.0(3.1)**	72.4(4.3)	70.3(3.9)	68.9(3.5)	71.3(3.5)	71.8(3.4)
04	**91.5(1.2)**	91.3(1.5)	89.6(1.7)	90.9(1.2)	89.4(1.2)	89.5(1.5)	**92.3(0.6)**	92.1(1.1)	91.2(1.2)	91.3(1.0)	90.2(1.5)	91.2(1.4)	**92.2(0.7)**	92.0(1.1)	91.1(1.1)	91.4(1.0)	90.2(1.5)	91.1(1.3)
05	**74.5(3.2)**	73.6(3.2)	69.8(3.8)	69.0(3.8)	68.1(2.6)	69.6(3.4)	**75.3(3.1)**	74.7(3.5)	69.5(2.9)	69.0(3.8)	68.6(2.3)	69.9(3.0)	**75.0(3.2)**	74.2(3.9)	69.4(2.5)	69.0(3.8)	68.5(3.0)	69.4(2.7)
06	69.4(2.9)	**69.9(2.1)**	65.3(2.5)	69.4(2.7)	64.8(1.9)	65.1(1.7)	72.8(3.1)	**73.6(2.9)**	67.9(2.2)	69.4(2.7)	66.4(1.8)	68.0(2.1)	72.6(3.5)	**73.3(2.8)**	67.5(1.8)	69.4(2.7)	66.2(2.4)	67.5(1.9)
07	71.2(3.3)	**71.9(4.4)**	67.5(2.6)	68.3(3.1)	67.3(2.6)	67.6(2.9)	**74.1(4.0)**	73.4(3.9)	67.7(2.6)	68.3(3.1)	67.0(3.0)	68.2(2.0)	**73.9(3.9)**	73.2(3.9)	67.5(2.6)	68.3(3.1)	66.6(3.1)	67.8(2.1)
08	69.8(5.6)	**72.1(3.6)**	71.1(4.8)	70.2(5.0)	70.7(4.1)	71.8(4.4)	**70.7(4.9)**	68.9(5.5)	65.5(5.5)	67.3(4.9)	66.2(5.2)	67.2(5.0)	**69.4(4.5)**	68.0(5.4)	64.5(4.7)	66.8(5.3)	65.8(5.3)	65.9(6.2)
09	74.3(3.8)	**79.9(6.1)**	73.8(4.0)	73.5(5.1)	72.5(5.9)	74.8(5.6)	78.1(3.8)	**79.3(5.9)**	75.5(3.7)	73.5(5.1)	71.9(5.2)	74.3(5.6)	77.6(3.8)	**78.5(5.5)**	74.7(3.9)	73.5(5.1)	72.0(5.0)	73.8(5.6)
10	69.4(4.3)	**69.9(3.9)**	66.8(4.3)	65.8(3.9)	67.9(4.2)	65.4(4.1)	70.5(4.6)	**72.1(3.7)**	65.8(3.7)	65.8(3.9)	67.3(3.7)	65.2(3.4)	69.6(4.5)	**72.1(4.0)**	65.5(3.1)	65.8(3.9)	67.5(3.7)	65.4(3.1)
11	86.5(3.2)	85.7(2.9)	84.3(2.8)	**87.4(4.5)**	84.7(4.0)	84.8(4.0)	89.9(2.3)	**90.0(1.9)**	87.9(3.0)	87.4(4.5)	87.1(3.5)	86.9(2.9)	89.9(2.4)	**90.0(1.9)**	88.3(3.1)	87.4(4.5)	87.2(3.5)	86.6(2.7)
12	**81.9(5.5)**	80.3(4.4)	80.1(4.8)	77.8(4.7)	79.8(4.6)	77.4(6.1)	**82.1(5.4)**	80.4(4.7)	77.3(5.4)	77.8(4.7)	77.3(4.9)	76.3(6.1)	**82.6(5.6)**	80.5(4.6)	77.5(6.2)	77.8(4.7)	76.3(4.6)	76.7(6.2)
13	72.0(4.1)	**72.2(4.8)**	66.0(4.1)	63.2(4.6)	67.8(4.7)	68.5(4.5)	69.3(5.8)	**72.4(4.2)**	63.4(5.6)	63.2(4.6)	63.7(4.9)	64.9(3.9)	69.0(6.1)	**72.0(4.8)**	62.0(5.6)	63.2(4.6)	63.4(5.1)	64.0(4.6)
14	95.9(1.0)	**96.6(0.9)**	96.1(0.8)	95.6(0.8)	96.2(0.8)	95.8(1.0)	96.1(0.8)	**96.5(0.8)**	96.3(0.9)	95.6(0.8)	95.8(0.9)	96.0(0.7)	95.9(0.8)	**96.4(0.8)**	96.1(0.9)	95.6(0.8)	95.7(0.9)	95.8(0.5)
15	**63.0(3.9)**	60.3(7.8)	55.6(5.7)	60.2(4.1)	55.0(5.6)	55.5(4.0)	**70.2(4.1)**	68.4(5.2)	61.3(4.3)	60.2(4.1)	63.1(4.6)	62.6(4.6)	**69.7(4.5)**	68.1(5.5)	60.8(4.9)	60.2(4.1)	61.7(4.4)	62.8(4.3)
16	**82.0(2.9)**	81.0(2.7)	80.0(2.3)	74.6(3.7)	79.5(2.3)	79.2(3.6)	**81.0(3.6)**	80.2(3.7)	77.2(3.4)	77.6(2.7)	78.4(2.9)	78.2(2.6)	**80.3(3.2)**	79.2(3.7)	76.0(3.4)	74.5(2.9)	78.0(2.8)	77.8(2.1)
17	99.9(0.4)	99.0(1.5)	98.2(2.4)	**100.0(0)**	93.0(3.5)	97.5(3.4)	**100.0(0)**	99.5(0.7)	99.2(1.4)	**100.0(0)**	97.5(2.2)	99.5(1.0)	**100.0(0)**	99.5(0.7)	99.2(1.4)	**100.0(0)**	97.5(2.2)	99.5(1.0)
18	90.3(1.5)	89.6(1.1)	88.2(1.3)	**91.7(1.7)**	88.4(1.8)	81.5(1.0)	**93.3(0.9)**	92.7(1.1)	91.5(1.0)	91.7(1.7)	91.2(1.3)	84.5(1.1)	**93.2(0.9)**	92.8(1.1)	91.6(1.0)	91.7(1.7)	91.2(1.3)	84.6(1.1)
19	84.2(2.0)	84.4(1.7)	81.9(1.1)	84.8(1.1)	82.8(1.1)	**87.8(1.5)**	86.6(1.5)	86.9(1.5)	84.6(0.9)	84.8(1.1)	85.0(0.9)	**92.0(1.2)**	86.6(1.5)	86.9(1.4)	84.8(0.9)	84.8(1.1)	85.0(0.8)	**92.1(1.2)**
20	94.7(1.2)	94.5(1.0)	93.4(0.9)	**95.3(1.1)**	92.4(1.2)	93.6(1.0)	**95.6(1.0)**	**95.6(0.9)**	95.2(1.2)	95.3(1.1)	93.8(1.4)	94.9(1.2)	**95.6(1.0)**	**95.6(0.9)**	95.1(1.2)	95.3(1.1)	93.9(1.4)	94.9(1.2)
21	70.0(7.4)	**70.8(6.4)**	64.9(8.4)	69.4(6.8)	63.7(7.1)	61.7(7.5)	74.7(7.2)	**75.8(5.0)**	71.1(6.3)	69.4(6.8)	70.6(7.0)	72.0(5.4)	74.5(6.8)	**75.7(5.0)**	71.4(6.3)	69.4(6.8)	70.8(7.0)	72.4(4.4)
22	**95.6(1.6)**	94.1(1.7)	95.3(1.2)	95.1(0.9)	94.6(1.2)	94.5(1.4)	95.4(1.3)	**95.8(1.8)**	94.8(1.6)	95.1(0.9)	94.5(1.4)	94.5(1.4)	95.3(1.3)	**95.8(1.7)**	94.8(1.6)	95.1(0.9)	94.4(1.5)	94.3(1.4)
23	**70.9(2.8)**	68.1(3.8)	64.2(3.9)	68.3(2.5)	64.3(3.6)	65.4(3.7)	**74.7(3.6)**	73.5(3.2)	68.4(2.4)	68.3(2.5)	67.5(2.0)	70.0(4.3)	**75.0(3.2)**	73.0(3.2)	68.5(3.5)	68.3(2.5)	67.4(1.8)	70.2(4.3)
24	**83.6(3.4)**	82.4(4.2)	79.8(4.4)	83.4(3.3)	78.2(4.6)	79.9(4.0)	**85.5(3.1)**	84.5(4.0)	82.8(3.3)	83.4(3.3)	81.1(4.0)	83.5(3.6)	**86.1(2.6)**	84.4(4.0)	82.1(4.0)	83.4(3.3)	81.3(4.2)	82.9(3.1)
25	**94.4(1.9)**	92.2(1.7)	92.1(1.9)	92.3(1.9)	91.3(2.3)	92.0(2.2)	94.5(2.4)	**94.6(1.5)**	93.0(2.0)	92.3(1.9)	92.5(2.2)	92.2(2.4)	**94.6(2.4)**	**94.6(1.6)**	93.0(2.1)	92.3(1.9)	92.4(2.2)	92.3(2.3)
26	**79.3(1.8)**	78.0(2.2)	74.9(1.3)	74.6(1.1)	74.1(1.2)	75.2(1.1)	**78.8(1.7)**	77.8(2.7)	74.4(1.3)	74.6(1.1)	73.0(1.2)	73.9(0.9)	**78.7(1.8)**	77.7(2.7)	74.4(1.2)	74.6(1.1)	72.9(1.2)	73.7(1.0)
27	78.9(6.2)	**79.8(5.7)**	73.4(6.3)	77.1(5.5)	74.7(7.5)	74.9(6.7)	**82.9(4.7)**	82.8(4.0)	77.9(4.6)	77.1(5.5)	77.8(3.4)	78.1(4.2)	82.7(4.4)	**82.9(3.8)**	78.0(4.6)	77.1(5.5)	78.0(3.0)	76.7(4.7)
28	**94.2(3.6)**	93.4(3.5)	91.7(4.7)	89.4(5.9)	90.6(5.2)	90.1(5.1)	**93.6(4.0)**	92.4(3.2)	91.5(4.5)	89.4(5.9)	89.9(4.8)	91.2(4.6)	**93.6(4.0)**	92.4(3.2)	91.5(4.5)	89.4(5.9)	89.9(4.8)	91.2(4.6)
Aver.	**82.0**	81.7	79.3	79.9	78.8	79.0	**83.4**	**83.2**	80.1	79.9	79.5	80.2	**83.2**	82.9	79.8	79.7	79.3	80.0

Table A.13
Comparison of DES methods using different pools of classifiers with Perceptron as base inducer (accuracy and Std. Dev.).

Data	Knora-E						Knora-U						Meta-Des					
	PGDCS_A	PGDCS_D	Bagging	AdaB.	R.F.	DSOC	PGDCS_A	PGDCS_D	Bagging	AdaB.	R.F.	DSOC	PGDCS_A	PGDCS_D	Bagging	AdaB.	R.F.	DSOC
01	82.3(3.1)	82.3(2.8)	82.2(3.1)	82.4(3.2)	**84.0(2.3)**	82.8(3.6)	87.5(2.3)	**88.1(2.0)**	86.9(2.3)	86.8(1.9)	88.0(2.0)	85.9(1.9)	86.2(2.7)	86.0(2.6)	86.2(2.5)	84.0(3.3)	**88.0(2.5)**	84.7(2.2)
02	96.9(0.8)	96.9(0.6)	96.9(0.7)	97.0(0.8)	**97.1(0.8)**	96.6(0.9)	96.1(1.0)	96.2(0.9)	96.2(1.0)	96.2(1.0)	**97.3(0.8)**	95.6(0.8)	94.4(1.1)	94.9(1.4)	93.3(1.2)	93.7(1.6)	**97.2(0.8)**	92.7(2.0)
03	70.3(4.1)	70.1(3.3)	70.2(3.8)	69.9(3.5)	**72.1(3.4)**	70.7(4.5)	78.3(2.4)	78.0(1.7)	**78.7(1.6)**	78.5(1.6)	75.1(2.1)	76.8(2.4)	76.0(1.1)	**76.2(1.1)**	76.0(0.9)	76.1(0.9)	75.8(1.4)	76.0(0.9)
04	89.8(1.5)	90.2(1.3)	89.5(1.6)	89.1(2.1)	**93.5(0.9)**	89.4(1.8)	89.5(1.0)	89.7(1.2)	89.0(2.1)	88.7(3.0)	**93.7(1.0)**	89.0(1.9)	89.4(1.2)	89.6(1.2)	88.8(1.1)	88.4(1.3)	**93.9(1.0)**	88.7(1.3)
05	**73.1(2.4)**	72.3(2.5)	72.5(2.7)	71.8(2.9)	71.7(3.0)	71.6(2.7)	**77.0(2.7)**	**77.0(2.8)**	76.9(2.7)	76.2(2.8)	75.8(2.8)	76.0(3.0)	74.5(3.3)	74.4(2.6)	73.7(2.5)	73.1(2.4)	**74.8(3.0)**	74.1(2.0)
06	68.6(2.0)	**69.9(1.4)**	68.9(1.7)	69.1(1.7)	69.8(1.8)	68.9(1.6)	72.3(1.6)	72.4(1.3)	71.7(1.5)	72.1(1.3)	**76.3(1.7)**	69.6(1.7)	70.8(1.3)	70.8(1.9)	70.3(1.4)	70.1(1.5)	**76.5(1.9)**	69.2(2.0)
07	71.0(1.5)	71.5(2.2)	**72.4(2.3)**	70.3(2.0)	71.1(1.8)	71.0(2.9)	**76.9(2.4)**	76.7(2.3)	76.1(2.4)	75.2(1.8)	75.3(2.4)	74.8(2.7)	73.0(2.0)	**73.8(1.8)**	72.9(1.7)	71.6(1.4)	72.9(1.8)	72.4(1.6)
08	67.2(4.8)	66.4(5.6)	65.6(6.1)	66.1(4.5)	**68.2(4.7)**	67.2(5.8)	**76.4(3.3)**	75.7(2.2)	75.1(2.1)	74.9(2.0)	70.5(3.7)	75.0(2.7)	73.4(1.6)	73.5(1.7)	**74.0(1.5)**	73.7(1.6)	72.0(2.6)	73.8(2.1)
09	**80.9(4.6)**	78.9(4.3)	79.4(4.2)	76.3(5.0)	76.0(5.3)	79.0(4.4)	**85.5(4.2)**	85.1(3.4)	82.8(3.8)	83.0(4.6)	80.1(5.3)	82.6(2.6)	**84.6(3.6)**	83.1(3.9)	82.9(4.0)	79.3(3.9)	79.9(3.9)	81.9(3.5)
10	67.9(3.0)	68.8(2.8)	68.9(2.3)	68.7(3.1)	**69.6(3.5)**	68.0(3.2)	70.9(2.4)	**72.4(2.1)**	71.0(2.4)	70.3(2.7)	71.0(3.2)	69.8(2.6)	70.7(2.2)	70.1(2.1)	70.7(1.6)	70.7(1.5)	**70.8(1.9)**	70.7(1.9)
11	88.8(3.7)	90.0(3.3)	88.4(3.6)	85.9(3.3)	**91.9(3.4)**	88.5(3.6)	90.7(3.1)	91.5(3.8)	89.2(2.8)	88.2(3.5)	**93.4(2.7)**	89.0(3.2)	90.3(2.7)	89.8(3.2)	88.8(3.0)	87.1(3.3)	**93.4(2.9)**	88.9(2.9)
12	**82.5(3.9)**	**82.5(4.4)**	79.9(3.7)	78.7(5.2)	79.7(4.4)	82.4(4.3)	**85.6(4.9)**	84.1(3.7)	83.0(3.9)	83.6(4.5)	82.8(4.5)	84.1(4.9)	**84.5(4.4)**	84.3(5.6)	83.0(5.5)	80.9(5.2)	83.1(4.4)	82.8(4.7)
13	66.5(4.6)	**68.5(4.5)**	66.7(4.0)	66.3(5.4)	67.7(5.2)	67.0(4.8)	75.1(3.0)	**75.9(2.6)**	73.5(2.1)	72.7(2.8)	73.0(2.8)	73.0(3.6)	71.2(3.0)	**72.1(3.2)**	71.1(2.8)	70.9(3.4)	70.2(4.1)	70.1(2.2)
14	95.8(0.7)	96.0(0.7)	96.0(0.7)	95.9(0.9)	**96.7(0.8)**	95.8(0.8)	95.0(1.2)	94.8(1.1)	95.1(1.1)	95.2(0.9)	**97.0(0.6)**	94.4(1.4)	93.9(1.4)	94.1(1.0)	94.2(1.1)	94.5(1.1)	**97.0(0.6)**	93.6(1.3)
15	65.6(4.9)	66.2(4.6)	**67.0(4.1)**	63.6(4.9)	63.3(4.0)	**67.0(4.2)**	**72.1(3.8)**	70.1(5.2)	70.1(3.5)	68.1(4.4)	69.0(4.6)	67.6(4.6)	67.7(4.8)	**68.2(4.1)**	67.2(4.7)	64.7(4.9)	68.0(4.4)	66.5(4.4)
16	77.4(2.8)	77.6(2.9)	77.2(2.7)	76.9(2.7)	**79.1(2.6)**	78.0(2.8)	**83.3(2.3)**	**83.3(2.3)**	82.9(1.9)	82.1(2.3)	79.3(2.0)	82.5(2.4)	79.9(3.3)	79.6(3.3)	79.7(3.0)	79.1(3.1)	79.2(2.3)	**80.1(3.5)**
17	89.4(3.1)	89.5(2.9)	88.9(3.1)	87.1(3.4)	**99.6(0.8)**	88.1(2.6)	87.3(1.9)	84.6(2.7)	81.8(3.5)	82.9(3.4)	**99.0(1.4)**	82.0(3.5)	94.4(3.2)	94.1(2.9)	91.5(4.5)	89.9(3.5)	**99.6(0.9)**	90.2(3.0)
18	**91.1(1.3)**	83.5(0.8)	83.5(0.9)	83.4(0.7)	88.3(0.8)	83.7(0.8)	**89.5(1.3)**	81.9(0.9)	81.8(1.0)	81.6(0.9)	**89.5(0.7)**	80.9(1.1)	88.7(2.2)	82.4(0.9)	82.5(0.9)	82.5(0.9)	**90.2(0.8)**	82.0(0.8)
19	83.1(0.8)	89.5(2.2)	90.5(1.6)	90.8(1.6)	**93.7(0.8)**	91.8(1.1)	81.7(0.9)	85.8(2.3)	87.2(2.8)	90.3(1.2)	**94.0(0.9)**	89.0(1.6)	82.4(1.0)	86.8(2.6)	86.8(2.5)	88.7(2.1)	**94.6(0.9)**	88.9(2.5)
20	94.0(1.1)	94.2(1.0)	94.2(1.0)	94.5(0.9)	**97.5(0.8)**	93.8(1.1)	91.9(1.0)	92.4(1.4)	92.3(1.0)	92.9(1.2)	**97.1(1.0)**	91.8(1.0)	94.0(1.0)	94.3(0.9)	94.0(1.1)	94.3(0.9)	**97.6(0.8)**	93.8(1.0)
21	**83.2(6.3)**	82.2(4.5)	78.5(3.5)	77.3(5.5)	77.9(5.0)	78.7(5.2)	**84.8(6.3)**	82.1(6.4)	78.1(3.0)	74.8(4.1)	78.7(4.0)	76.5(3.6)	**84.9(5.3)**	82.7(6.0)	79.7(3.7)	76.6(5.7)	80.2(3.9)	78.8(3.8)
22	96.2(1.6)	**96.3(1.5)**	95.9(1.8)	95.6(1.5)	96.2(0.8)	96.0(1.5)	97.2(1.0)	**97.4(0.9)**	97.0(1.0)	97.0(1.1)	96.1(0.8)	96.6(1.3)	**97.0(0.9)**	96.7(0.8)	96.4(0.9)	96.4(1.0)	95.5(1.3)	96.4(0.9)
23	75.5(2.5)	**75.8(2.6)**	**75.8(2.2)**	74.8(2.2)	71.0(2.1)	74.8(2.4)	76.5(2.9)	**77.7(2.1)**	76.0(2.1)	76.8(2.6)	74.0(2.2)	75.1(2.4)	76.2(1.9)	**76.6(2.4)**	75.9(2.3)	75.9(2.7)	74.5(1.6)	74.9(2.7)
24	**85.2(3.9)**	85.1(3.5)	84.8(4.3)	83.1(4.4)	84.1(4.1)	84.9(3.8)	**88.0(4.2)**	87.1(3.5)	86.5(4.0)	86.0(3.5)	85.8(3.0)	86.1(4.1)	**88.0(3.0)**	87.3(3.0)	86.7(3.8)	86.1(3.3)	85.1(4.3)	86.7(3.6)
25	97.4(1.0)	**97.7(1.2)**	97.1(0.9)	96.3(1.3)	95.4(1.6)	97.1(1.1)	98.1(0.7)	**98.2(1.0)**	97.1(1.2)	96.7(1.3)	95.5(1.5)	97.0(1.0)	97.2(1.2)	**97.5(1.6)**	96.9(1.3)	96.0(1.3)	95.7(1.4)	96.7(1.4)
26	83.7(1.0)	**83.8(1.0)**	83.4(0.8)	83.6(1.0)	80.3(0.8)	83.3(1.2)	**86.8(0.9)**	**86.8(1.0)**	86.4(0.8)	86.6(0.8)	85.1(0.8)	85.8(0.9)	**86.1(0.9)**	85.9(0.9)	85.9(0.8)	85.8(0.7)	84.8(0.9)	85.6(0.8)
27	82.9(3.8)	81.9(4.7)	82.1(4.6)	75.9(5.5)	**84.9(3.2)**	80.5(4.5)	85.2(3.8)	85.5(3.7)	83.2(3.6)	82.5(4.5)	**89.7(2.7)**	81.8(3.6)	85.3(3.4)	84.4(4.1)	82.7(3.4)	81.1(4.8)	**89.1(3.6)**	80.8(4.4)
28	**98.4(1.9)**	97.3(2.4)	97.5(2.0)	97.2(2.6)	96.6(2.8)	97.2(1.6)	**98.5(1.8)**	98.0(2.3)	97.5(2.1)	97.2(2.6)	97.2(3.0)	97.4(1.7)	**98.4(1.9)**	98.2(2.2)	98.0(2.0)	97.3(2.2)	97.2(2.7)	97.2(2.0)
Aver.	82.3	82.3	81.9	81.0	**82.8**	81.9	84.9	84.6	83.7	83.5	**85.0**	83.1	84.0	83.8	83.2	82.4	**84.9**	82.8

Table A.14
Comparison of DES methods using different pools of classifiers with Decision Tree as base inducer (accuracy and Std. Dev.).

Data	Knora-E						Knora-U						Meta-Des					
	PGDCS_A	PGDCS_D	Bagging	AdaB.	R.F.	DSOC	PGDCS_A	PGDCS_D	Bagging	AdaB.	R.F.	DSOC	PGDCS_A	PGDCS_D	Bagging	AdaB.	R.F.	DSOC
01	**88.6(2.7)**	88.1(2.9)	84.9(3.0)	82.3(2.6)	84.0(2.3)	84.2(2.2)	**92.4(2.5)**	91.1(3.0)	87.6(2.1)	82.3(2.6)	88.0(2.0)	86.8(1.9)	**91.7(2.4)**	90.4(2.5)	86.6(2.8)	82.3(2.6)	87.9(2.5)	85.8(2.0)
02	**97.6(0.7)**	97.5(0.6)	97.2(0.7)	95.8(0.8)	97.1(0.8)	96.9(0.7)	**97.9(0.7)**	97.7(0.7)	97.1(0.9)	95.8(0.8)	97.3(0.8)	97.0(0.9)	**97.9(0.7)**	97.6(0.7)	97.1(0.8)	95.8(0.8)	97.2(0.8)	96.9(0.9)
03	**75.8(3.4)**	75.0(4.1)	71.8(4.4)	69.8(3.7)	72.1(3.4)	72.8(3.5)	**82.7(2.1)**	80.8(2.7)	77.4(2.6)	76.1(2.2)	75.1(2.1)	76.3(3.5)	**79.1(1.8)**	78.8(2.1)	75.6(1.1)	76.0(1.0)	75.8(1.4)	75.7(1.3)
04	**94.7(1.2)**	94.6(1.1)	93.0(0.9)	91.9(1.4)	93.5(0.9)	92.9(0.9)	**95.7(1.0)**	95.4(1.0)	93.3(1.0)	92.0(1.5)	93.7(1.0)	92.9(0.8)	**95.6(0.9)**	95.3(0.8)	93.4(1.0)	92.0(1.5)	93.9(1.0)	93.1(0.9)
05	**79.0(2.7)**	77.0(4.1)	71.7(2.9)	69.0(3.8)	71.7(3.0)	72.0(3.0)	**85.0(2.8)**	82.1(4.5)	76.5(2.6)	69.0(3.8)	75.8(2.8)	75.6(3.2)	**84.8(2.8)**	81.5(5.1)	75.0(2.4)	69.0(3.8)	74.8(3.0)	74.3(3.0)
06	76.6(3.6)	**77.2(2.8)**	71.0(1.8)	69.4(2.7)	69.8(1.8)	69.9(1.7)	83.6(4.8)	**84.5(2.6)**	75.9(1.6)	69.4(2.7)	76.3(1.7)	74.5(2.0)	84.0(4.2)	**84.6(2.9)**	76.2(1.7)	69.4(2.7)	76.5(1.9)	74.6(1.8)
07	**79.5(3.7)**	77.8(4.4)	70.8(2.1)	68.3(3.1)	71.1(1.8)	71.0(2.3)	**85.5(5.1)**	81.8(5.3)	75.3(2.5)	68.3(3.1)	75.3(2.4)	74.0(3.2)	**85.1(4.7)**	81.0(5.2)	73.1(2.8)	69.7(1.7)	72.9(1.8)	72.4(2.3)
08	**73.2(5.0)**	71.6(4.8)	66.9(4.5)	69.4(4.9)	68.2(4.7)	68.0(5.3)	**80.3(5.9)**	79.9(4.9)	71.5(3.5)	71.8(4.3)	70.5(3.7)	71.6(4.5)	**78.4(5.3)**	77.4(4.8)	72.0(3.6)	73.2(1.3)	72.0(2.6)	71.8(2.8)
09	**82.2(4.5)**	80.2(4.1)	78.3(4.6)	73.5(5.1)	76.0(5.3)	75.6(4.7)	**89.8(3.8)**	87.8(4.7)	80.5(4.6)	73.5(5.1)	80.1(5.3)	79.7(5.7)	**88.2(4.1)**	85.1(4.5)	79.4(3.7)	73.5(5.1)	79.9(3.9)	78.3(5.7)
10	73.1(4.6)	**76.1(4.0)**	67.7(3.1)	65.8(3.9)	69.6(3.5)	67.4(3.5)	81.4(5.1)	**81.7(4.9)**	70.5(3.1)	65.8(3.9)	71.0(3.2)	69.7(3.3)	78.8(4.6)	**80.5(4.1)**	70.8(1.6)	71.2(0)	70.8(1.9)	70.3(2.1)
11	**95.7(2.0)**	94.9(2.5)	91.1(3.5)	87.4(4.5)	91.9(3.4)	90.4(3.3)	**96.2(1.5)**	96.0(2.0)	92.6(2.7)	87.4(4.5)	93.4(2.7)	92.0(2.9)	**96.4(1.6)**	96.0(1.9)	92.7(3.0)	87.4(4.5)	93.4(2.9)	92.2(2.2)
12	**85.5(5.5)**	84.1(3.8)	80.9(5.2)	77.8(4.7)	79.7(4.4)	79.5(6.1)	**91.0(4.9)**	88.1(5.8)	82.9(4.3)	77.8(4.7)	82.8(4.5)	82.7(4.8)	**90.7(5.3)**	87.7(5.3)	82.2(4.6)	77.8(4.7)	83.1(4.4)	81.8(5.7)
13	72.6(5.3)	**74.5(5.7)**	66.5(4.5)	63.2(4.6)	67.7(5.2)	66.6(4.0)	80.5(5.2)	**81.1(4.3)**	73.2(3.4)	63.2(4.6)	73.0(2.8)	71.1(3.2)	78.5(6.2)	**79.0(5.9)**	70.2(4.4)	67.9(3.9)	70.2(4.1)	69.0(3.8)
14	**97.1(0.6)**	**97.1(0.7)**	96.6(0.7)	95.6(0.8)	96.7(0.8)	96.5(0.8)	**97.9(0.7)**	97.7(0.5)	97.2(0.6)	95.6(0.8)	97.0(0.6)	96.8(0.6)	**97.7(0.7)**	97.5(0.6)	97.0(0.7)	95.6(0.8)	97.0(0.6)	96.8(0.6)
15	**73.7(4.5)**	70.7(5.5)	63.4(3.9)	60.2(4.1)	63.3(4.0)	64.8(5.0)	**84.5(3.5)**	79.1(7.1)	70.1(4.0)	60.2(4.1)	69.0(4.6)	68.3(3.7)	**83.7(4.1)**	78.7(6.1)	67.9(4.6)	59.8(4.1)	68.0(4.4)	66.4(5.2)
16	**81.7(3.0)**	80.1(2.9)	78.9(2.8)	76.4(3.3)	79.1(2.6)	79.6(2.6)	**85.1(4.0)**	84.3(3.4)	81.5(2.1)	74.5(4.5)	79.3(2.0)	80.3(2.0)	**83.2(2.9)**	82.2(3.2)	80.4(2.8)	79.3(2.1)	79.2(2.3)	79.9(2.9)
17	**100.0(0)**	**100.0(0)**	**100.0(0)**	**100.0(0)**	99.6(0.8)	93.5(1.3)	**100.0(0)**	**100.0(0)**	**100.0(0)**	**100.0(0)**	99.0(1.4)	99.6(1.1)	**100.0(0)**	**100.0(0)**	**100.0(0)**	**100.0(0)**	99.6(0.9)	**100.0(0)**
18	95.3(1.1)	95.0(1.0)	93.4(1.0)	91.7(1.7)	93.7(0.8)	**99.8(0.8)**	**96.9(1.3)**	95.9(1.1)	93.6(1.0)	91.7(1.7)	94.0(0.9)	93.0(1.2)	**96.5(1.1)**	96.0(1.0)	94.5(1.0)	91.7(1.7)	94.6(0.9)	88.7(0.8)
19	**90.2(2.1)**	**90.2(1.5)**	87.4(0.8)	84.8(1.1)	88.3(0.8)	86.8(0.7)	**92.1(2.5)**	92.0(1.9)	88.5(0.9)	84.8(1.1)	89.5(0.7)	87.5(0.8)	92.3(2.1)	92.6(1.5)	89.7(0.7)	84.8(1.1)	90.2(0.8)	**94.0(1.2)**
20	**98.0(1.0)**	97.9(0.8)	96.8(0.8)	95.3(1.1)	97.5(0.8)	96.4(0.9)	**98.1(0.9)**	97.9(0.9)	96.4(0.9)	95.3(1.1)	97.1(1.0)	96.0(1.0)	**98.3(1.1)**	98.2(0.8)	97.0(0.7)	95.3(1.1)	97.6(0.8)	96.6(1.0)
21	80.3(4.8)	**81.9(4.9)**	75.7(4.9)	69.4(6.8)	77.9(5.0)	75.2(4.6)	84.3(5.2)	**85.8(6.6)**	75.7(5.6)	69.4(6.8)	78.7(4.0)	73.2(5.8)	83.8(5.4)	**85.5(6.6)**	78.3(3.7)	69.4(6.8)	80.2(3.9)	76.1(4.5)
22	96.7(1.4)	**97.1(1.1)**	95.6(1.1)	95.1(0.9)	96.2(0.8)	95.4(1.3)	97.6(1.3)	**97.8(1.0)**	95.9(1.1)	95.1(0.9)	96.1(0.8)	95.8(1.0)	97.6(1.4)	**97.9(1.2)**	95.4(1.0)	94.7(1.4)	95.5(1.3)	95.5(0.9)
23	**79.5(3.4)**	77.5(3.9)	70.9(3.0)	68.3(2.5)	71.0(2.1)	72.1(3.8)	**85.6(3.1)**	84.9(5.1)	73.3(1.8)	68.3(2.5)	74.0(2.2)	73.2(3.2)	**84.7(3.0)**	83.7(6.0)	73.9(2.2)	68.3(2.5)	74.5(1.6)	73.4(3.0)
24	**88.0(3.7)**	87.1(2.7)	83.2(4.3)	83.4(3.3)	84.1(4.1)	83.7(3.1)	92.1(3.9)	**92.3(4.1)**	85.8(3.3)	83.4(3.3)	85.8(3.0)	84.5(3.5)	**92.1(4.4)**	91.8(4.1)	84.9(3.2)	83.4(3.3)	85.1(4.3)	84.1(3.8)
25	**97.0(1.2)**	96.2(1.4)	94.8(1.3)	92.3(1.9)	95.4(1.6)	94.8(1.2)	**97.4(1.8)**	96.6(1.7)	95.0(1.7)	92.3(1.9)	95.5(1.5)	94.5(2.0)	**97.4(1.7)**	96.8(1.2)	95.0(1.5)	92.3(1.9)	95.7(1.4)	94.5(1.7)
26	**86.3(2.2)**	85.2(3.1)	80.0(1.1)	74.6(1.1)	80.3(0.8)	78.7(1.0)	**90.5(2.5)**	89.6(2.9)	84.4(0.8)	74.6(1.1)	85.1(0.8)	83.3(0.9)	**90.3(2.6)**	89.5(2.9)	83.9(0.8)	74.6(1.1)	84.8(0.9)	82.9(0.8)
27	**88.9(4.2)**	86.9(4.3)	82.0(4.3)	77.1(5.5)	84.9(3.2)	79.3(3.8)	**93.5(3.3)**	90.3(3.1)	86.5(4.0)	77.1(5.5)	89.7(2.7)	83.7(3.5)	**93.3(3.1)**	92.3(3.5)	86.6(3.6)	77.1(5.5)	89.1(3.6)	84.1(4.1)
28	**98.6(2.0)**	96.4(2.7)	96.2(2.8)	89.4(5.9)	96.6(2.8)	95.5(3.4)	**99.4(1.0)**	96.5(2.5)	95.7(4.1)	89.4(5.9)	97.2(3.0)	94.9(3.8)	**99.1(1.3)**	96.2(2.7)	96.0(4.2)	89.4(5.9)	97.2(2.7)	94.7(4.0)
Aver.	**86.6**	86.0	82.4	79.9	82.8	82.1	**90.6**	89.6	84.8	80.1	85.0	83.9	**90.0**	89.1	84.5	80.7	84.9	83.7

Appendix. Detailed results

This section contains the detailed results of the two PGDCS variants (PGDCS_A, PGDCS_D) for all datasets using Perceptron and Decision Trees as base classifiers. Such results represent the average and standard deviation of 20 iterations.

Tables A.9 and A.10 show the results when all the classifiers in the pool were combined using Majority Voting Rule (MVR). Tables A.11 and A.12 present the results related to the use of Dynamic Classifier Selection (DCS). Finally, Tables A.13 and A.14 show the results related to the use of Dynamic Ensemble Selection.

References

[1] A.S. Britto Jr., R. Sabourin, L.E. Oliveira, Dynamic selection of classifiers—a comprehensive review, Pattern Recognit. 47 (11) (2014) 3665–3680.

[2] R.M. Cruz, R. Sabourin, G.D. Cavalcanti, Dynamic classifier selection: Recent advances and perspectives, Inf. Fusion 41 (2018) 195–216.

[3] L.I. Kuncheva, A theoretical study on six classifier fusion strategies, IEEE Trans. Pattern Anal. Mach. Intell. 24 (2002) 281–286.

[4] L. Breiman, Bagging predictors, Mach. Learn. 24 (2) (1996) 123–140.

[5] Y. Freund, R. Schapire, Experiments with a new boosting algorithm, Int. Conf. Mach. Learn. (1996) 148–156.

[6] T.K. Ho, The random subspace method for constructing decision forests, IEEE Trans. Pattern Anal. Mach. Intell. 20 (8) (1998) 832–844, http://dx.doi.org/10.1109/34.709601.

[7] L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.

[8] T.K. Ho, M. Basu, M.H. Law, Measures of geometrical complexity in classification problems, in: Data Complexity in Pat. Recog., Springer, 2006, pp. 1–23.

[9] D. Ruta, B. Gabrys, Classifier selection for majority voting, Inf. Fusion 6 (1) (2005) 63–81.

[10] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.

[11] M. Monteiro, A.S.B. Jr., J.P. Barddal, L.S. Oliveira, R. Sabourin, Classifier pool generation based on a two-level diversity approach, in: 25th International Conference on Pattern Recognition, ICPR 2020, Virtual Event / Milan, Italy, January 10-15, 2021, IEEE, 2020, pp. 2414–2421, http://dx.doi.org/10.1109/ICPR48806.2021.9412137.

[12] L.I. Kuncheva, Combining Pattern Classifiers: Methods and Algorithms, John Wiley & Sons, 2014.

[13] T.K. Ho, M. Basu, Complexity measures of supervised classification problems, IEEE Trans. Pattern Anal. Mach. Intell. 24 (3) (2002) 289–300, http://dx.doi.org/10.1109/34.990132.

[14] J. Luengo, F. Herrera, An automatic extraction method of the domains of competence for learning classifiers using data complexity measures, Knowl. Inf. Syst. 42 (1) (2015) 147–180.

[15] A.L. Brun, A.S. Britto, L.S. Oliveira, F. Enembreck, R. Sabourin, Contribution of data complexity features on dynamic classifier selection, in: Proc. Int. Jt. Conf. Neural Networks. 2016-Octob, (July) 2016, pp. 4396–4403.

[16] A.L. Brun, A.S. Britto Jr., L.S. Oliveira, F. Enembreck, R. Sabourin, A framework for dynamic classifier selection oriented by the classification problem difficulty, Pattern Recognit. 76 (2018) 175–190.

[17] A.C. Lorena, L.P.F. Garcia, J. Lehmann, M.C.P. Souto, T.K. Ho, How complex is your classification problem? A survey on measuring classification complexity, ACM Comput. Surv. 52 (5) (2019) http://dx.doi.org/10.1145/3347711.

[18] A. Orriols-Puig, N. Macì, T.K. Ho, Documentation for the Data Complexity Library in C++, Vol. 196, Universitat Ramon Llull, la Salle, 2010, pp. 1–40.

[19] R.A. Mollineda, J.S. Sánchez, J.M. Sotoca, Data characterization for effective prototype selection, in: Iberian Conf. on Pattern Recog. and Image Analysis, Springer, 2005, pp. 27–34, URL http://sci2s.ugr.es/keel/pdf/specific/congreso/lncs05bmollineda.pdf.

[20] W. Malina, Two-parameter Fisher criterion, IEEE Trans. Syst. Man Cybern. B 31 (4) (2001) 629–636.

[21] R.L. Graham, P. Hell, On the history of the minimum spanning tree problem, Ann. Hist. Comput. 7 (1) (1985) 43–57.

[22] R. Lileikyte, L. Telksnys, Quality estimation methodology of speech recognition features/snekos signalu atpazinimo pozymiu kokybes matas, Elektronika Ir Elektrotechnika (2011) 113–117.

[23] V.H. Barella, L.P. Garcia, M.P. de Souto, A.C. Lorena, A. De Carvalho, Data complexity measures for imbalanced classification tasks, in: Int. Joint Conf. on Neural Networks, IJCNN, IEEE, 2018, pp. 1–8.

[24] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[25] R.M.O. Cruz, L.G. Hafemann, R. Sabourin, G.D.C. Cavalcanti, DESlib: A dynamic ensemble selection library in Python, J. Mach. Learn. Res. 21 (8) (2020) 1–5.

[26] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, C. Gagné, DEAP: Evolutionary algorithms made easy, J. Mach. Learn. Res. 13 (2012) 2171–2175.

[27] M. Lichman, et al., UCI Machine Learning Reposit, Irvine, CA, 2013.

[28] R. Duin, P. Juszczak, P. Paclik, E. Pekalska, D. deRidder, D. Tax, S. Verzakov, Prtools, a matlab toolbox for pattern recog, 2004, URL http://www.prtools.org.

[29] R.D. King, C. Feng, A. Sutherland, Statlog: comparison of classification algorithms on large real-world problems, Appl. Artif. Intell. Int. J. 9 (3) (1995) 289–333.

[30] L. Kuncheva, Ludmila kuncheva collection LKC, 2004, Available: http://pages.bangor.ac.uk/mas00a/activi.

[31] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, F. Herrera, Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework, J. Mult. Valued Logic Soft Comput. 17 (2011).

[32] C. Jutten, The enhanced learning for evolutive neural architectures project, 2002.

[33] G. Valentini, An experimental bias-variance analysis of SVM ensembles based on resampling techniques, IEEE Trans. Syst. Man Cybern. B 35 (6) (2005) 1252–1271.

[34] E. Hernández-Reyes, J.A. Carrasco-Ochoa, J.F. Martínez-Trinidad, Classifier selection based on data complexity measures, in: Iberoamerican Congress on Pattern Recognition, Springer, 2005, pp. 586–592.

[35] E. Leyva, A. González, R. Perez, A set of complexity measures designed for applying meta-learning to instance selection, IEEE Trans. Knowl. Data Eng. 27 (2) (2014) 354–367.

[36] H. Jain, K. Deb, An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach, Part II: Handling constraints and extending to an adaptive approach, IEEE Trans. Evol. Comput. 18 (4) (2014) 602–622, http://dx.doi.org/10.1109/TEVC.2013.2281534, URL http://ieeexplore.ieee.org/document/6595567/.

[37] K. Woods, W.P. Kegelmeyer, K. Bowyer, Combination of multiple classifiers using local accuracy estimates, IEEE Trans. Pattern Anal. Mach. Intell. 19 (4) (1997) 405–410.

[38] A.H. Ko, R. Sabourin, J. Alceu Souza Britto, From dynamic classifier selection to dynamic ensemble selection, Pattern Recognit. 41 (5) (2008) 1718–1731.

[39] R.M. Cruz, R. Sabourin, G.D. Cavalcanti, T. Ing Ren, META-DES: A dynamic ensemble selection framework using meta-learning, Pattern Recognit. 48 (5) (2015) 1925–1935.

[40] R. Cruz, R. Sabourin, G. Cavalcanti, META-DES. Oracle: meta-learning and feature selection for dynamic ensemble selection, Inform. Fusion (38) (2017) 84–103.

[41] R.M. Cruz, R. Sabourin, G.D. Cavalcanti, A DEEP analysis of the META-DES framework for dynamic selection of ensemble of classifiers, 2015, arXiv preprint arXiv:1509.00825.

[42] J. Demšar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (Jan) (2006) 1–30.