Paper accepted for International Journal of Bio-Inspired Computation, 2014

A Differential Evolution Algorithm to Optimise the Combination of Classifier and Cluster Ensembles

Luiz F. S. Coletta*, Eduardo R. Hruschka*, Ayan Acharya† and Joydeep Ghosh†

*Department of Computer Science
University of Sao Paulo (USP)
Sao Carlos, SP, Brazil
{luizfsc,erh}@icmc.usp.br

†Department of Electrical & Computer Engineering,
University of Texas (UT),
Austin, TX, USA
aacharya@utexas.edu, ghosh@ece.utexas.edu

June 6, 2014

Abstract: Unsupervised models can provide supplementary soft constraints to help classify new data since similar instances are more likely to share the same class label. In this context, this paper reports on a study on how to make an existing algorithm, named C^{3}E (from Consensus between Classification and Clustering Ensembles), more convenient by automatically tuning its main parameters. The C^{3}E algorithm is based on a general optimisation framework that takes as input class membership estimates from existing classifiers, and a similarity matrix from a cluster ensemble operating solely on the new (target) data to be classified, in order to yield a consensus labeling of the new data. To do so, two parameters have to be defined *a priori* by the user: the relative importance of classifier and cluster ensembles, and the number of iterations of the algorithm. We propose a Differential Evolution (DE) algorithm, named Dynamic DE (D^{2}E), which is a computationally efficient alternative for optimising such parameters. D^{2}E provides better results than DE by dynamically updating its control parameters. Moreover, competitive results were achieved when comparing D^{2}E with three state-of-the-art algorithms.

1 Introduction

The combination of multiple classifiers to generate a single classifier has been an active area of research over the past two decades (Kuncheva, 2004; Kittler and Roli, 2002). For instance, an analytical framework that quantifies the improvements in classification results due to the combination of multiple models was addressed in Tumer and Ghosh (1996). More recently, a survey into ensemble techniques — including their applications to many difficult real-world problems, such as remote sensing, person recognition, one vs. all recognition, and medicine — was reported in Oza and Tumer (2008). The literature on the subject has shown that from independent and diversified classifiers, the ensemble created is usually more accurate than its individual components. Analogously, just as ensemble learning has been proved more useful than single-model solutions to classification problems, several research efforts have shown that cluster ensembles can improve the quality of results in comparison to a single clustering solution (Ghosh and Acharya, 2011).

This paper investigates the use of a metaheuristic to estimate values for user-defined parameters of an algorithm that combines ensembles of classifiers and clusterers. Most of the motivations for combining ensembles of classifiers and clusterers are similar to those that hold for the standalone use of either classifier ensembles or cluster ensembles. However, some additional nice properties can emerge from such a combination — *e.g.*, unsupervised models can provide a variety of supplementary constraints for classifying new (target) data (Basu et al., 2008). From this viewpoint, the underlying assumption is that similar new instances in the target set are more likely to share the same class label. Thus, the supplementary constraints provided by the cluster ensemble can be useful for improving the generalization capability of the resulting classifier.

Acharya et al. (2011) introduced a framework that combines ensembles of classifiers and clusterers to generate a more consolidated classification. In this framework, an ensemble of classifiers is first learned on an initial labeled training dataset. Such classifiers are then used to obtain initial estimates of class probability distributions for new unlabeled (target) data. In addition, a cluster ensemble is applied to the target data to yield a similarity matrix that is used to refine the initial class probability distributions obtained from the classifier ensemble. This framework is materialized through an optimisation algorithm that exploits properties of Breg-

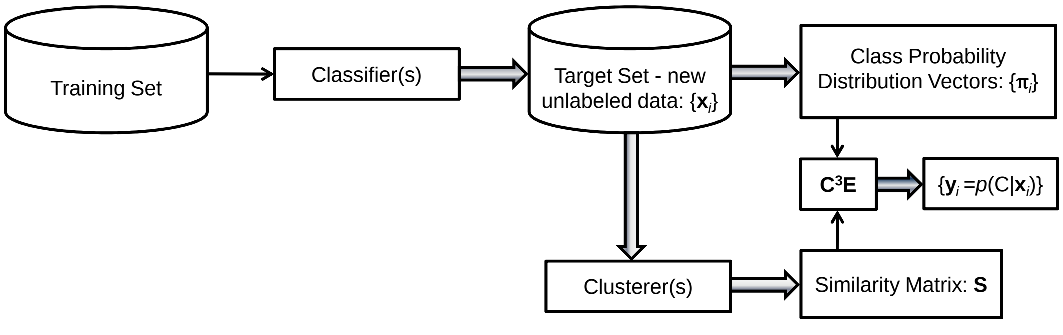
Training Set
Classifier(s)
Target Set - new unlabeled data: {x_i}
Class Probability Distribution Vectors: {π_i}
C³E
{y_i = p(C|x_i)}
Clusterer(s)
Similarity Matrix: S

Figure 1: Framework to combine ensembles of classifiers and clusterers using C^{3}E algorithm (Acharya et al., 2011).

man divergences[1] in conjunction with Legendre duality to yield a principled and scalable approach. As discussed in Section 2.1, by using a squared loss function, the algorithm becomes simpler and requires only two user-defined parameters — as opposed to the more general version of the algorithm, which has three user-defined parameters. The optimisation of such parameters by means of a Differential Evolution (DE) algorithm, named Dynamic DE (D^{2}E), is the main contribution of our work.

The remainder of the paper is organized as follows. Section 2 reviews the combination of classifier and cluster ensembles as performed by the C^{3}E algorithm, with particular emphasis on its simpler version based on a squared loss function, which was not explicitly studied in Acharya et al. (2011). Section 3 describes the Differential Evolution (DE) algorithm (Price et al., 2005; Storn and Price, 1997; Price, 1996) and introduces the D^{2}E algorithm, which is more robust to control parameters variations. Section 4 provides our experimental analysis, including comparisons of D^{2}E with three state-of-the-art DE variants. Finally, Section 5 concludes the paper.

2 Combination of classifier and cluster ensembles

Acharya et al. (2011) designed a framework that combines classifiers and clusterers to generate a more consolidated classification. This framework, whose core is the C^{3}E algorithm, is depicted in Figure 1. It is assumed that a set of classifiers (consisting of one or more classifiers) has been previously induced from a training set. Such an ensemble of classifiers is employed to estimate initial class probability distributions for every instance $\mathbf{x}_i$ of a target set $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^n$. These probability distributions are stored as c-dimensional vectors $\{\boldsymbol{\pi}_i\}_{i=1}^n$ (c is the number of classes) and will be refined with the help of a cluster ensemble. From this viewpoint, the cluster ensemble provides supplementary constraints for the classification of the instances of $\mathcal{X}$, with the rationale that similar instances are more likely to share the same class label. In order to capture similarities between the instances of $\mathcal{X}$, C^{3}E takes as input a similarity (co-association) matrix $\mathbf{S}$, where each entry corresponds to the relative co-occurrence of two instances in the same cluster (Ghosh and Acharya, 2011; Strehl and Ghosh, 2002) — considering all the data partitions that form the cluster ensemble built from $\mathcal{X}$.

To sum up, C^{3}E receives as inputs a set of vectors $\{\boldsymbol{\pi}_i\}_{i=1}^n$ and the matrix $\mathbf{S}$, and outputs a consolidated classification for every instance in $\mathcal{X}$ — represented by a set of vectors $\{\mathbf{y}_i\}_{i=1}^n$, where $\mathbf{y}_i = p(C \mid \mathbf{x}_i)$ — i.e., $\mathbf{y}_i$ is the estimated posterior class probability assignment for every instance in $\mathcal{X}$. To do so, C^{3}E solves an optimisation problem whose objective is to minimize J in (1) with respect to the set of probability vectors $\{\mathbf{y}_i\}_{i=1}^n$:

$$J = \sum_{i \in \mathcal{X}} \mathcal{L}(\mathbf{y}_i, \boldsymbol{\pi}_i) + \alpha \sum_{(i,j) \in \mathcal{X}} s_{ij} \mathcal{L}(\mathbf{y}_i, \mathbf{y}_j) \qquad (1)$$

Quantity $\mathcal{L}(\cdot,\cdot)$ denotes a loss function. Informally, the first term in Equation (1) captures dissimilarities between the class probabilities provided by the ensemble of classifiers and the output vectors $\{\mathbf{y}_i\}_{i=1}^n$. The second term encodes the cumulative weighted dissimilarity between all possible pairs $(\mathbf{y}_i, \mathbf{y}_j)$. The weights are assigned to these pairs proportionally to the similarity values $s_{ij} \in [0,1]$ of matrix $\mathbf{S}$ and coefficient $\alpha \in \mathbb{R}_+$ controls the relative importance of classifier and cluster ensembles.

The C^{3}E algorithm, as proposed in Acharya et al. (2011), exploits general properties of a large class of loss functions, described by Bregman divergences (Banerjee et al., 2005), in conjunction with Legendre duality and a notion of variable splitting also used in alternating direction method of multipliers (Boyd et al., 2011) to yield a principled and scalable solution. If a squared loss function is chosen, the C^{3}E algorithm becomes simpler. In particular, the variable splitting approach, in which two copies of $\mathbf{y}_i$ are updated iteratively, is no longer necessary. As such, we can get rid of one of the user-defined parameters of the algorithm, λ, which is an optimisation constraint to ensure that the two copies of the variables remain close during the optimisation process. The update equations are still available in closed form solution for each $\mathbf{y}_i$, as explained as follows.

2.1 C^{3}E with Squared Loss Function

By choosing the Squared Loss (SL) function as the Bregman divergence in the optimisation problem formulated in Equation (1) we obtain:

[1] Bregman Divergences include a large number of useful loss functions such as the well-known Squared Loss, KL-divergence, Mahalanobis Distance, and I-Divergence.

$$J_{SL} = \frac{1}{2}\sum_{i \in \mathcal{X}} \|\mathbf{y}_i - \boldsymbol{\pi}_i\|^2 + \alpha \frac{1}{2} \sum_{(i,j) \in \mathcal{X}} s_{ij} \|\mathbf{y}_i - \mathbf{y}_j\|^2 \quad (2)$$

The second term of J_{SL} shows that the variables are coupled. In order to circumvent this difficulty, and following an approach analogous to the one adopted for the more general case (Acharya et al., 2011) — in which any Bregman Divergence can be used — we can design an iterative update procedure of the variables to be optimised. In particular, keeping $\{\mathbf{y}_j\}_{j=1}^n \setminus \{\mathbf{y}_i\}$ fixed, we can minimize J_{SL} in Equation (2) for every $\mathbf{y}_i$ by setting:

$$\frac{\partial J_{SL}}{\partial \mathbf{y}_i} = \mathbf{0}. \quad (3)$$

Considering that the similarity matrix $\mathbf{S}$ is symmetric and observing that $\frac{\partial \|\mathbf{x}\|2}{\partial \mathbf{x}} = 2\mathbf{x}$, we obtain:

$$\mathbf{y}_i = \frac{\boldsymbol{\pi}_i + \alpha' \sum_{j \neq i} s_{ij} \mathbf{y}_j}{1 + \alpha' \sum_{j \neq i} s_{ij}}, \quad (4)$$

where $\alpha' = 2\alpha$ has been set for mathematical convenience. Equation (4) can be computed iteratively by using Algorithm 1, which summarizes the main steps of C^{3}E with Squared Loss function. Since each update of $\mathbf{y}_i$ reduces J_{SL}, which is bounded from below by zero, the algorithm converges. A stopping criterion can be defined as either the maximum number of iterations, I, or a predefined threshold on the difference between values of the objective function in (2) computed from two consecutive iterations of the algorithm.

The asymptotic time complexity of the algorithm is $O(c \cdot n^2)$, where c is the number of class labels and n is the number of instances in the target set. Note that n, the number of new instances to be classified, is usually much smaller than the number of instances in the training set used to build the classifier ensemble. Furthermore, as in the more general case addressed in Acharya et al. (2011), the resulting minimization procedure can be performed in parallel by updating one or more variables per processor.

In practice, the choice of the Bregman divergence is dependent on the application domain. At this point, however, we can anticipate that our empirical results are very similar to those found in Acharya et al. (2011) across a variety of datasets, suggesting that, as usual, simpler approaches should be tried first. In this sense, we have investigated how to automatically optimise the user-defined parameters of the algorithm based on the Squared Loss function (C^{3}E-SL), namely coefficient α, which controls the relative importance of classifier and cluster ensembles, and the number of iterations (I) of the Algorithm 1.

As expected, this is a difficult (multi-modal) optimisation problem (with several local maxima), as shown in Figure 2 — which reports the different classification accuracies by varying α and I for the *Wine Red Quality* dataset (Frank and Asuncion, 2010). We are interested in high accuracies, which are represented by multiple peaks spread in specific regions of the search space. It is widely known that metaheuristics, such as those based on evolutionary algorithms, have been successfully used in this type of problem (Yang, 2010; Eiben and Smith, 2008). This is our initial motivation to study the use of Differential Evolution (DE) algorithms to optimise the parameters of C^{3}E-SL and, in particular, develop the D^{2}E algorithm.

Algorithm 1: C^{3}E with Squared Loss – C^{3}E-SL

```
  Input: $\{\boldsymbol{\pi}_i\}, \mathbf{S}, \alpha$.
  Output: $\{\mathbf{y}_i\}$.
1 Initialize $\{\mathbf{y}_i\}$ such that $y_{i\ell} = \frac{1}{c}$ $\forall i \in \{1, 2, ..., n\}$, $\forall \ell \in \{1, 2, ..., c\}$;
2 Repeat
3     Update $\mathbf{y}_i$ using Equation (4) $\forall i \in \{1, 2, ..., n\}$;
4 until convergence;
```

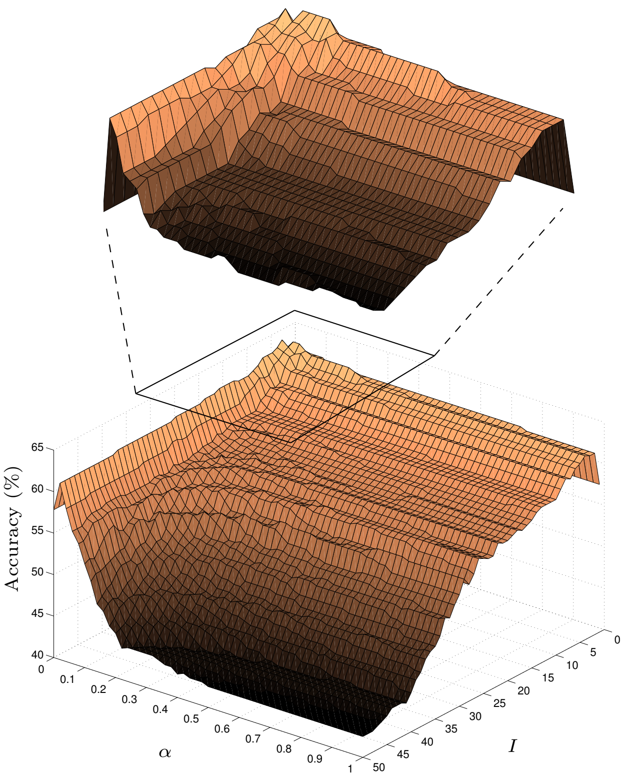

Figure 2: C^{3}E-SL classification accuracy (%) with respect to different combinations of values for parameters α and I.

3 A differential evolution algorithm to optimise the C^{3}E-SL parameters

A variety of metaheuristics have been designed to address problems that cannot be suitably tackled through more traditional optimisation algorithms — see (Yu et al., 2013; Zhong et al., 2012; Yang and Deb, 2012; Garcia-Gonzalo and Fernandez-Martinez, 2012; Natarajan et al., 2012; Ali and Sabat, 2012; Abdelaziz et al., 2012). In multi-modal optimisation problems, such as those tackled when optimising the C^{3}E-SL parameters, it is of particular interest to use algorithms capable of escaping from local optima, hopefully being able to reach the global optimum — or at least local optimum solutions (within a reasonable computation time). Reaching good solutions in less time is particularly suitable for data mining applications that typically operate on large time-demanding data (Madden, 2012; Kantardzic, 2011; Freitas, 2002).

Essentially, metaheuristics consist of stochastic algorithms with randomization and local search that are used for global optimisation, including evolutionary-based algorithms (Bäck et al., 1991; Bäck and Schwefel, 1993) and those based on thermodynamic principles (Kirkpatrick et al., 1983) and swarm intelligence (Kennedy and Eberhart, 1995). In general, metaheuristics have two main characteristics: diversification and intensification (Yang, 2010). Diversification involves exploring the search space on a global scale by generating diverse random solutions, whereas intensification focuses on the search in a specific region of the search space by exploiting good solutions found in it. Several studies have shown that metaheuristics are capable of finding acceptable solutions for complex (optimisation) problems in reasonable time, preventing entrapment in local minima (Eiben and Smith, 2008; Bäck et al., 1997; Fleming and Purshouse, 2002; Freitas, 2003). In particular, DE-based algorithms have proved simple and very effective for parameter optimisation (Das and Suganthan, 2011; Price et al., 2005), therefore, we decided to investigate them to optimise the C^{3}E-SL parameters.

3.1 A brief review of Differential Evolution

Differential Evolution (DE) (Price et al., 2005; Storn and Price, 1997; Price, 1996) aims at solving an optimisation problem by maintaining a population of D-dimensional candidate solutions (parameter vectors). From this population, new candidate solutions are created by means of perturbations (the so-called mutations) of existing solutions. In particular, consider a given parameter vector. After being mutated, the parameters of this vector are mixed with the parameters of another predetermined (target) vector to provide the so-called trial vector. More specifically, let Np be the number of individuals in the population, then for each target vector $\mathbf{x}_{i,G}$ of generation G, $i = 1, 2, ..., Np$, a mutant vector is generated according to:

$$\mathbf{v}_{i,G+1} = \mathbf{x}_{r_1,G} + F \cdot (\mathbf{x}_{r_2,G} - \mathbf{x}_{r_3,G}) \quad (5)$$

where r_1, r_2, $r_3 \in \{1, 2, ..., Np\}$ are mutually different random indexes and F is a scaling constant. Then, a trial vector $\mathbf{u}_{i,G+1} = (u_{1i,G+1}, u_{2i,G+1}, \ ..., u_{Di,G+1})$ is produced by the crossover operator, according to the following rules:

$$u_{ji,G+1} = \begin{cases} v_{ji,G+1}, & \text{if } (d_1(j) \leq Cr) \text{ or} \\ & \quad (j = d_2(i)) \\ x_{ji,G}, & \text{if } (d_1(j) > Cr) \text{ and} \\ & \quad (j \neq d_2(i)) \end{cases} \quad (6)$$

where $d_1(j)$ is the jth output of a random number generator that uses a uniform distribution of numbers in [0,1], $d_2(i)$ is a randomly chosen index $\in \{1, 2, ..., D\}$ which ensures that $\mathbf{u}_{i,G+1}$ gets at least one parameter from $\mathbf{v}_{i,G+1}$, and Cr is the crossover constant. If the trial vector $\mathbf{u}_{i,G+1}$ encodes a better solution than the target vector $\mathbf{x}_{i,G}$, this gets replaced by the trial vector $\mathbf{u}_{i,G+1}$ in the generation $G+1$; otherwise, the old vector $\mathbf{x}_{i,G}$ is retained.

DE can be materialized by means of a variety of trial vector generation strategies. To classify different DE strategies, the notation "DE/x/y/z" is used (Price et al., 2005), where x specifies the vector to be mutated — e.g., *rand* (a randomly chosen population vector) or *best* (the best vector from the current population); y represents the number of difference vectors used; and z denotes the type of crossover — *bin* or *exp*. Thus, the formulation described above can be written as "DE/rand/1/bin".

As one can observe, DE has three user-defined control parameters, namely number of individuals in the population (Np), scaling constant (F), and crossover constant (Cr). According to Price (1996) and Storn and Price (1997), $F = 0.5$ and $Cr = 0.9$ are good initial attempts to set up the algorithm. If the user wishes to make a fine tuning to improve the results, the suggested values are $F \in [0.4, 1]$ and $Cr \in [0.8, 1]$ (Storn, 1996; Storn and Price, 1997; Ronkkonen et al., 2005). In this sense, recent studies have shown that the performance of DE depends on the correct choice of the control parameters (Ronkkonen et al., 2005; Gamperle et al., 2002; Liu and Lampinen, 2002). Inappropriate values for F and Cr may lead to slow convergence and/or low accuracies[2].

Over the last years, many authors have developed techniques for automatically setting the DE control parameters (Das and Suganthan, 2011; Wang et al., 2011; Qin et al., 2009; Brest et al., 2006). These techniques can involve deterministic, adaptive, and self-adaptive parameter control approaches (Eiben and Smith, 2008). The proposed D^{2}E can be considered an algorithm with adaptive parameter control, so that new settings are determined by taking into account some feedback from the search process.

3.2 Dynamic Differential Evolution (D^{2}E)

Based on trial vector generation strategies discussed in Section 3.1, D^{2}E can be written as "DE/best/2/bin" (Price, 1996), in which Equation (5) is replaced by:

$$\begin{aligned} \mathbf{v}_{i,G+1} \;\; = \;\; & \mathbf{x}_{best,G} + \\ & F \cdot (\mathbf{x}_{r_1,G} + \mathbf{x}_{r_2,G} - \mathbf{x}_{r_3,G} - \mathbf{x}_{r_4,G}). \end{aligned} \quad (7)$$

Essentially, D^{2}E extends DE by sampling values for F and Cr when, in two consecutive generations, there are no changes in the average fitness of the current population. To do so, the algorithm uses the following rules:

[2]The population size and number of generations required for convergence are somewhat related. From a practical viewpoint, one may expect that the more individuals, the fewer the required generations for convergence.

$$<F, Cr>_{G+1} = \begin{cases} <d_3, d_4>, & \text{if } (\bar{f}_G = \bar{f}_{G-1}) \\ <F, Cr>_G, & \text{otherwise} \end{cases} \quad (8)$$

where $d_3 \in [0.1, 1]$, $d_4 \in [0, 1]$ are values randomly chosen from a uniform distribution, $\bar{f}_G$ and $\bar{f}_{G-1}$ are the average fitness of the population in generation G and $G-1$, respectively. These dynamic updates of the parameters can prevent the algorithm from getting stuck in local minima and speeding up the convergence (as shown in Section 4.2).

D^{2}E starts with control parameters provided by the user (e.g., $Np = 20$, $F = 0.5$, and $Cr = 0.9$). As explained, it allows F and Cr to assume new values if there is no improvement in two consecutive generations. Therefore, D^{2}E can avoid local minima (with sufficiently high values of F and/or Cr), as well as it can emphasize exploitation (with sufficiently low values of F and/or Cr). From this perspective, the impact of the initial user-defined control parameters on the results decreases. In other words, this adaptation provides a good exploration-exploitation trade-off, making the algorithm more robust with respect to initial values of both F and Cr.

The next section provides experimental results that show that D^{2}E can efficiently optimise the C^{3}E-SL parameters to build good classifiers for a variety of datasets. In particular, D^{2}E is compared with DE and three state-of-the-art algorithms: Self-adaptive DE — SaDE (Qin et al., 2009), jDE (Brest et al., 2006), and Composite DE — CoDE (Wang et al., 2011).

4 Empirical evaluation

The optimisation of the parameters of the C^{3}E-SL algorithm (described in Section 2.1) consists in searching for values of α and I that yield the best classification accuracy. In other words, ideally we are looking for an optimal pair of values $<\alpha^*, I^*>$ that results in the most accurate classifier for a given problem. In order to estimate such values, we employ the algorithm addressed in Section 3.2. The details of our experimental setup, followed by the empirical results, are presented next.

4.1 Experimental setup

Ten datasets from the UCI Machine Learning Repository (Frank and Asuncion, 2010) were used in the experiments. The main characteristics of these datasets are shown in Table 1.

Table 1: Main characteristics of the used datasets.

ID	Dataset	Inst.	Attrib.	Classes
Wisc	Wisconsin Breast Cancer	683	9	2
Pima	Pima Indians Diabetes	768	8	2
Yeast	Yeast	205	20	4
Iono	Ionosphere	351	33	2
Iris	Iris	150	4	3
Blood	Blood Transfusion Service Center	748	4	2
Seeds	Seeds	210	7	3
Ecoli	Ecoli	336	7	8
Ilpd	Indian Liver Patient	579	10	2
Glass	Glass Identification	214	9	7

Essentially, the optimal values for the parameters of C^{3}E-SL — $<\alpha^*, I^*>$ — can be estimated based on cross-validation procedures, in which three datasets are usually employed: training, validation, and test. In a controlled experimental setting, the new data (target set) is frequently referred to as either a test or a validation set. These two terms have been used interchangeably in the literature, sometimes causing confusion. In our study we have assumed that the target/test set has not been used in the process of building the ensemble of classifiers — *i.e.*, it is an independent set of data not used at all to optimise any parameter of the algorithm and the resulting classifier. Thus, as usual (Witten and Frank, 2005), only the training and validation sets are used to optimise the parameters of the algorithm.

Figure 3 illustrates the two main steps of the adopted optimisation procedure. First, we split the instances into training, validation, and target/test sets. The ensemble of classifiers is built with the training set. Then, we use a DE-based algorithm to estimate α^* and I^* by utilizing the validation set, where a cluster ensemble is induced. The resulting values for the parameters are finally employed to assess the classification accuracy in the target/test set where, again and as requested by C^{3}E, a cluster ensemble similar to the one built for the validation set must be induced. Provided that, in a controlled experimental setting, we do know the true class labels of all instances, thus we can repeat this process multiple times and compute statistics of interest from the target/test set. We used the cross-validation procedure for empirically estimating the generalization capability of the C^{3}E-SL algorithm (as further discussed next).

A straightforward, but often computationally intensive, way of searching for $<\alpha^*, I^*>$ involves running grid-search (Bergstra and Bengio, 2012), which has been adopted as a baseline for comparison purposes in our work. Grid search usually refers to an exhaustive search through a subset of the parameter space. In our case, one of the parameters, i.e., number of iterations (I), is nat-

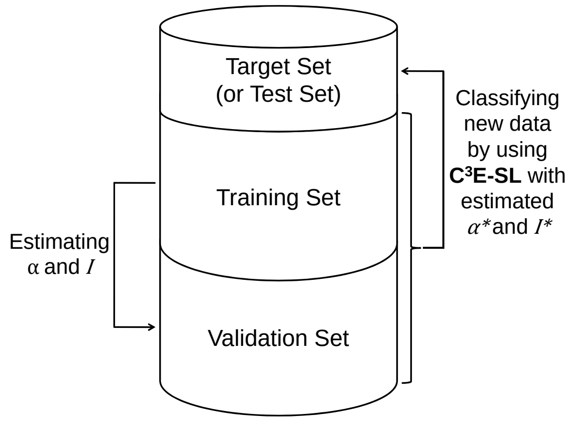

Figure 3: Optimising C^{3}E-SL parameters via cross-validation.

Algorithm 2: Optimising the C^{3}E-SL parameters by means of metaheuristics

```
  Input: MH ← choose a metaheuristic.
  Output: estimated pair of values $<\alpha^*, I^*>$.
1 Run MH:
2     Randomly set initial solution(s) for $<\alpha, I>$ such
      that $\alpha = \{0, 0.001, ..., 1\}$ and $I = \{1, 2, ..., 50\}$ —
      a population containing a certain number of
      individuals (solutions) is initialized;
3     Run C$^3$E-SL (Algorithm 1) for each individual
      of the population (each one using its coded
      solution $<\alpha, I>$ as input for the C$^3$E-SL) — the
      misclassification rate, as in Equation (9), is used
      to determine the goodness of the individuals;
4     Apply the MH operators;
5     Go to Step 3;
6 until time limit is reached;
7 Select the best individual (which provides the
  minimal misclassification rate) to be the pair of
  values $<\alpha^*, I^*>$.
```

urally discrete, whereas the other parameter (α) is real-valued, requiring discretization before grid-search running. For both, the lower and upper bounds must be set *a priori*. In particular, we used all different combinations of $\alpha = \{0, 0.001, 0.002, ..., 1\}$ and $I = \{1, 2, 3, ..., 50\}$. From this setting, C^{3}E-SL was run 50,500 times for estimating the optimal pair of values $<\alpha^*, I^*>$ for each dataset, according to:

$$<\alpha^*, I^*> = \underset{\alpha, I}{arg\ min}\, \text{ClassErr}\left[\text{C}^3\text{E-SL}(<\alpha, I>)\right] . \quad (9)$$

D^{2}E was run to minimize the misclassification rate in (9) from a time constraint given by grid-search. For each dataset, the running times spent by the grid-search were stored and then 50% of them were given as a time limit for D^{2}E. Algorithm 2 provides an overview of the general framework in which metaheuristics, as DE-based algorithms, can be employed to optimise the C^{3}E-SL parameters[3]. Accuracy results from D^{2}E were then compared to those found via grid-search. So that we could evaluate the trade-off between running time and classification accuracy provided by the algorithm. It is worth emphasizing that C^{3}E-SL optimised by means of D^{2}E reaches good accuracies in less running time and without the use of user-defined critical parameters. This is the main benefit of our work (as addressed in Section 4.2).

We adopted a 5-fold cross-validation process (Witten and Frank, 2005), in which each fold has 20% of the dataset instances — this is precisely the size of each target/test set. Accordingly, both training and validation sets contain 40% of the dataset instances. The classifier ensemble was composed of two well-known classifiers, namely Naive Bayes and Decision Tree (Witten and Frank, 2005). The similarity matrix **S** was constructed from a cluster ensemble based on data partitions found by the K-Means clustering algorithm (Jain, 2010): one with k clusters and another with $2k$ clusters — where the number of clusters, k, was automatically estimated from data by the relative validity clustering criterion known as Simplified Silhouette (Campello and Hruschka, 2006; Hruschka et al., 2006).

[3]We shall note that the DE algorithms used in this work were implemented in Matlab using only the necessary commands. The population initialization was carried on within a common region (of α and I), so that, more uniform (efficiency) comparisons could be performed. The same computer (i7, 3.2 GHz, 12 Gb RAM) running only the operational system was used for all the controlled experiments.

4.2 Experimental results

Before reporting the experimental results achieved by D^{2}E, it is instructive to compare the results obtained by C^{3}E based on Squared Loss (SL) with C^{3}E based on I-Divergence — following Acharya et al. (2011) we adopted $\lambda = 1$. For both variants of C^{3}E, grid-search was performed to optimise their parameters — α and I (as addressed in Section 4.1). Table 2 shows the average classification accuracies (from 5-fold cross-validation), along with the results obtained for the classifier ensembles.

In most of the cases both variants of C^{3}E (using grid search) provided better classification accuracies in comparison to the classifier ensemble (composed of Naive Bayes and Decision Tree). The overall accuracies of C^{3}E-SL were also similar to those of C^{3}E based on I-Divergence, which is more complex.

A sensitivity analysis was carried out in order to investigate the behavior of DE-based algorithms to variations of their parameters. Table 3 shows the average accuracies (from 5-fold cross-validation) obtained by DE with different combinations of $F = \{0.25, 0.5, 0.75, 1\}$ and $Cr = \{0, 0.25, 0.5, 0.75, 1\}$, along with initial suggestions of the literature (Price, 1996; Storn and Price, 1997), which are $F = 0.5$ and $Cr = 0.9$ (in the last column). The strategy "DE/best/2/bin" was used to minimize the misclassification rate in (9) in half the running time of the grid-search (by using Algorithm 2). As expected, the best settings for F and Cr were not the same for different problems. However, a more careful observation indicates — as pointed by Ronkkonen et al. (2005); Storn

Table 2: Average accuracies (%) of Classifier Ensemble, C^{3}E based on I-Divergence (Acharya et al., 2011) (ID) and C^{3}E based on Squared Loss (SL). Standard deviations appear within parentheses. For convenience, the best results are highlighted in bold face.

Dataset	**Class. Ens.**	**C^{3}E-ID**	**C^{3}E-SL**
Wisc	95.60 (2.8)	**96.48** (1.2)	**96.48** (1.2)
Pima	76.18 (3.1)	**77.09** (3.6)	75.92 (2.7)
Yeast	95.61 (3.2)	96.59 (2.8)	**97.56** (1.7)
Iono	**88.34** (4.3)	85.20 (11.2)	87.75 (7.1)
Iris	95.33 (3.0)	**96.00** (2.8)	**96.00** (2.8)
Blood	75.80 (1.9)	76.47 (1.5)	**76.60** (1.7)
Seeds	88.57 (7.4)	90.00 (3.1)	**91.90** (3.2)
Ecoli	**84.83** (3.8)	**84.83** (4.5)	84.53 (4.8)
Ilpd	62.69 (3.4)	**66.84** (9.4)	60.98 (8.2)
Glass	**64.97** (6.7)	62.65 (9.9)	**64.97** (6.7)

Table 3: Average accuracies (%) of C^{3}E-SL optimised by DE with different combinations of F and Cr (which remained fixed). For convenience, the best results are highlighted in bold face — LIT refers to parameter values suggested by the literature (Price, 1996; Storn and Price, 1997), i.e., $F = 0.5$ and $Cr = 0.9$.

F	0.25					0.50					0.75					1.00					**LIT**
Cr	0.00	0.25	0.50	0.75	1.00	0.00	0.25	0.50	0.75	1.00	0.00	0.25	0.50	0.75	1.00	0.00	0.25	0.50	0.75	1.00	
Wisc	95.8	96.5	95.8	96.2	95.9	95.8	96.0	96.2	96.0	96.5	95.9	96.0	96.2	96.5	96.5	95.8	96.0	96.2	95.9	96.6	**96.8**
Pima	71.6	**76.4**	74.8	72.7	73.5	74.1	**76.4**	73.7	75.4	75.0	72.7	**76.4**	75.3	75.3	**76.4**	74.2	74.5	76.0	75.5	75.0	75.0
Yeast	95.6	95.6	96.6	96.6	95.1	96.6	94.6	**98.0**	95.6	96.6	96.1	97.6	97.6	97.6	97.6	94.6	97.6	97.6	96.1	97.6	97.6
Iono	86.9	81.5	85.5	86.6	82.6	84.9	85.2	82.0	86.9	87.2	83.2	86.0	86.3	87.2	85.2	82.3	85.2	86.9	86.3	85.7	**89.2**
Iris	95.3	**96.0**	94.7	**96.0**	**96.0**	90.0	90.0	95.3	90.0	**96.0**	90.0	**96.0**	90.0	**96.0**	95.3	90.0	**96.0**	**96.0**	95.3	**96.0**	95.3
Blood	76.3	75.7	**76.6**	76.3	76.3	76.3	76.2	**76.6**	75.4	75.8	76.3	75.4	75.4	76.5	76.2	75.5	76.2	75.4	76.3	76.2	75.8
Seeds	89.0	89.0	89.5	88.6	87.1	89.0	88.6	89.5	88.6	89.0	89.5	89.0	89.0	88.6	89.0	88.6	89.5	89.5	89.0	**91.0**	89.1
Ecoli	83.6	79.8	83.3	84.8	83.6	84.2	83.0	**85.1**	**85.1**	83.9	77.4	84.2	83.0	84.2	83.6	83.3	84.5	**85.1**	83.0	83.9	84.2
Ilpd	61.3	61.7	61.3	60.6	61.3	61.0	60.5	61.3	61.3	61.7	60.5	**62.2**	60.8	61.1	60.5	61.3	61.3	**62.2**	61.1	61.7	61.3
Glass	59.4	61.7	**65.0**	63.6	61.2	56.6	60.8	62.2	62.6	63.6	59.4	60.8	59.8	62.6	63.6	62.6	60.8	**65.0**	**65.0**	64.5	**65.0**

Table 4: Average accuracies (%) of C^{3}E-SL optimised by D^{2}E with different combinations of initial values of F and Cr (which were dynamically updated). For convenience, the best results are highlighted in bold face — LIT refers to parameter values suggested by the literature (Price, 1996; Storn and Price, 1997), i.e., $F = 0.5$ and $Cr = 0.9$.

F	0.25					0.50					0.75					1.00					**LIT**
Cr	0.00	0.25	0.50	0.75	1.00	0.00	0.25	0.50	0.75	1.00	0.00	0.25	0.50	0.75	1.00	0.00	0.25	0.50	0.75	1.00	
Wisc	95.8	96.5	96.2	95.9	96.5	96.2	**96.8**	**96.8**	96.0	96.3	96.5	96.0	95.8	96.2	96.2	96.3	96.0	96.3	96.5	**96.8**	96.5
Pima	73.8	**76.3**	74.0	72.9	74.6	75.9	74.9	72.7	75.1	74.9	75.0	74.5	75.8	75.0	**76.3**	74.6	75.9	76.2	75.3	76.2	**76.3**
Yeast	97.1	97.1	97.6	95.6	97.1	97.6	97.6	**98.1**	97.6	96.1	95.6	**98.1**	95.6	97.1	96.1	97.6	97.1	**98.1**	97.1	97.1	97.1
Iono	86.6	87.2	87.2	86.0	84.3	87.2	86.3	87.2	86.3	86.3	86.3	86.3	82.9	86.0	86.3	81.8	86.3	82.6	84.3	85.7	**89.2**
Iris	**96.0**	**96.0**	90.0	**96.0**	**96.0**	90.0	**96.0**	90.0	**96.0**	**96.0**	**96.0**	90.0	**96.0**	**96.0**	**96.0**	**96.0**	95.3	95.3	**96.0**	90.0	**96.0**
Blood	75.4	76.1	76.2	76.2	76.2	75.3	76.5	76.2	76.1	**76.7**	76.5	75.4	76.1	75.8	75.8	76.2	75.4	76.5	75.4	75.4	76.1
Seeds	86.7	89.5	89.1	89.1	87.1	89.1	89.1	**91.0**	89.1	89.1	89.5	86.7	86.7	89.1	89.1	89.5	89.1	89.5	89.1	89.1	89.1
Ecoli	**85.1**	83.6	84.2	83.6	84.5	84.2	84.5	84.2	84.8	84.8	83.6	**85.1**	84.5	**85.1**	83.9	83.3	83.9	83.3	83.9	83.6	83.9
Ilpd	60.3	62.0	61.3	61.1	60.5	60.8	60.5	61.7	61.3	61.3	61.1	61.7	61.5	60.8	61.1	61.1	60.5	61.3	61.7	**62.5**	60.5
Glass	64.0	**64.5**	63.1	63.6	61.3	62.2	59.8	61.7	63.1	61.3	63.6	62.2	62.2	63.1	63.1	**64.5**	60.8	61.3	62.2	63.6	59.8
$\geq$	6	9	5	5	8	8	8	5	7	4	9	6	6	4	5	7	4	5	6	4	5

and Price (1997) and Storn (1996) — that $F \in [0.4, 1]$ and $Cr \in [0.8, 1]$ provide good results. Table 4, shows the average accuracies obtained by D^{2}E. Initial values of F and Cr were dynamically updated according to (8). The last row denotes the number of datasets where D^{2}E obtained equal or higher classification accuracies than DE (for the same values of F and Cr).

In order to provide some reassurance about the validity and non-randomness of the results, we performed statistical tests by following the approach described by Demšar (2006). In short, this approach is aimed at comparing multiple algorithms in multiple datasets by using the well-known Friedman test with a corresponding post-hoc test. This statistical procedure indicates that there is no statistically significant difference when comparing the accuracies obtained by grid-search, DE and D^{2}E. We can conclude that DE and D^{2}E provided classifiers as accurate as those obtained from grid-search, but taking half the running time. Considering the DE-based algorithms only, D^{2}E is preferable because it holds the nice properties of DE, while offering robustness to the initial values of its parameters. As a result, with the initial values $F = Cr = 0.25$, D^{2}E provided equal or superior results than DE in 90% of the datasets — this same percentage is reached for $F = 0.75$ and $Cr = 0.0$.

Paired comparisons were also made looking at each dataset individually — then considering, for each one, the different combinations of F and Cr in Tables 3 and 4. Figure 4 summarizes these comparisons by showing the proportions of equal or higher accuracies of D^{2}E over DE across the datasets. By observing this figure, one can note that D^{2}E achieved the best results in most of the cases — particularly, for *Wisconsin, Iris, Seeds,* and *Ecoli.* In addition, Figure 5 illustrates, for each dataset, the standard deviations of DE algorithms computed from twenty-one different accuracies in Tables 3 and 4 — specific values can be found in Table 5. These results show that, in most of the datasets, D^{2}E is less sensitive to initial parameter variations, providing (as a result) higher and more stable classification accuracies.

Figures 6–11 illustrate the average misclassification rates during the search performed by DE and D^{2}E in two datasets (*Wisconsin* and *Blood*) according to their respective values of F and Cr. These figures are typical, and have been obtained for a particular fold of the

Table 5: Standard deviations of DE and D^{2}E from twenty-one different settings of their control parameters. Lower values are highlighted in bold face.

Dataset	**DE**	**D^{2}E**
Wisc	0.31	**0.30**
Pima	1.36	**1.07**
Yeast	1.07	**0.78**
Iono	2.00	**1.74**
Iris	2.66	**2.59**
Blood	**0.42**	0.44
Seeds	**0.69**	1.09
Ecoli	1.80	**0.58**
Ilpd	**0.49**	0.57
Glass	2.27	**1.39**

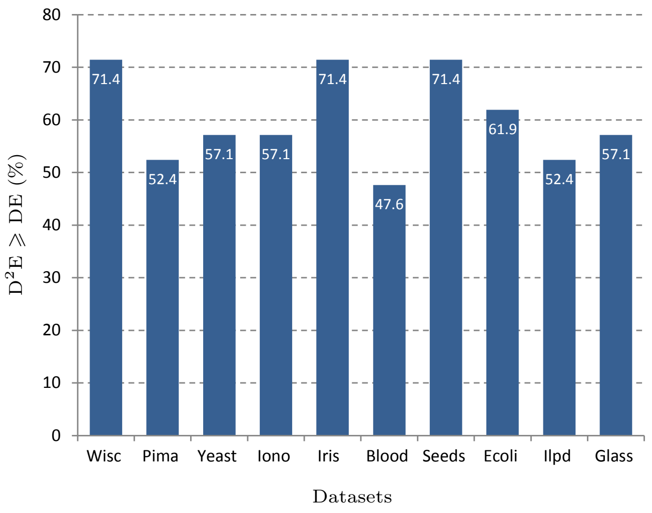

Figure 4: Proportions of equal or higher accuracies of D^2E over DE from twenty-one different settings of control parameters.

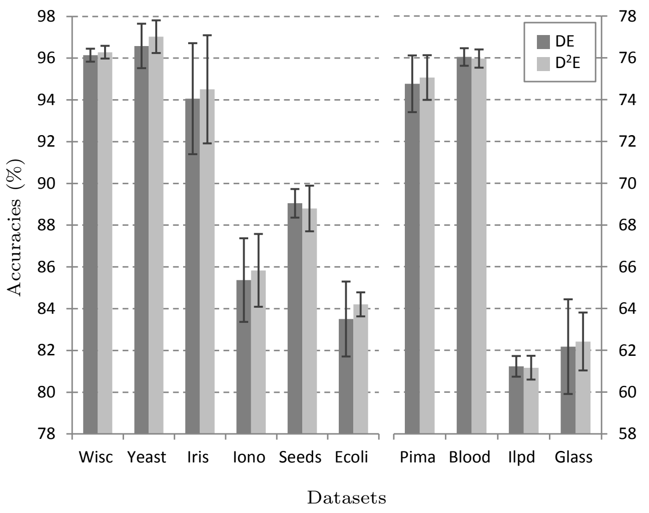

Figure 5: Average accuracies with standard deviations of DE and D^2E from twenty-one different settings of control parameters.

cross-validation procedure. For DE, values of F and Cr remained fixed as $F = 0.5$ and $Cr = 0.75$, whereas for D^2E they were dynamically updated. Figures 6 and 9 show that these updates, from the fifteenth generation on, contributed to a faster decrease in the misclassification rates. Such a behavior occurs in most of the datasets. Considering the *Wisconsin* dataset, from the fourth generation on we can note that new (lower) values for F and Cr helped to enhance the fitness — see Figures 6–8. The results for *Blood* (Figures 9–11) show that we are dealing with a difficult optimisation problem and by varying F and Cr, better results can be achieved.

4.3 Comparison with state-of-the-art algorithms

D^2E was compared with three state-of-the-art DE variants, namely: Self-adaptive DE — SaDE (Qin et al., 2009), jDE (Brest et al., 2006), and Composite DE — CoDE (Wang et al., 2011). These algorithms, such as D^2E, adapt their control parameters (and the trial vector generation strategy) along the evolutionary process. The settings used for each one are those recommended in the literature (Wang et al., 2011; Qin et al., 2009; Brest et al., 2006).

Experiments were performed according to Section 4.1. Thus, DE variants were used with Algorithm 2 aiming at minimizing the misclassification rate in a validation set — in half of the running time of the grid-search — as shown in Figure 3 (we adopted 5-fold cross-validation).

Table 6 reports the average classification accuracies provided by C^3E-SL optimised by each DE variant — D^2E results were obtained by setting $F = Cr = 0.25$. The last three rows summarize the number of datasets where D^2E average accuracy is superior, equal, and inferior to the corresponding algorithm, respectively. According to these results, we can state that D^2E showed competitive results to the three state-of-the-art algorithms — there is no statistically significant difference among them according to the Friedman test.

Table 6: Average accuracies (%) of C^3E-SL optimised by DE variants (standard deviations in parentheses). Win/Tie/Loss denote that the average accuracy of D^2E is superior, equal, and inferior to the corresponding algorithm — the best results are highlighted in bold face.

Dataset	**SaDE**	**jDE**	**CoDE**	**D^2E**
Wisc	**96.85** (0.5)	96.82 (0.4)	96.82 (0.6)	96.48 (1.2)
Pima	74.15 (1.5)	74.06 (1.6)	74.06 (1.6)	**76.31** (2.7)
Yeast	96.59 (2.7)	96.71 (2.8)	96.71 (2.1)	**97.07** (2.0)
Iono	87.39 (4.6)	86.68 (3.0)	**88.53** (3.2)	87.17 (9.3)
Iris	**96.00** (2.8)	95.67 (3.1)	**96.00** (2.8)	**96.00** (2.8)
Blood	76.87 (0.7)	**77.01** (0.7)	**77.01** (0.7)	76.07 (0.5)
Seeds	**91.43** (2.7)	90.83 (2.3)	87.14 (2.2)	89.52 (2.1)
Ecoli	83.41 (2.6)	83.56 (2.8)	83.33 (2.8)	**83.64** (5.4)
Ilpd	64.12 (6.2)	62.00 (5.1)	**67.23** (2.3)	62.01 (7.2)
Glass	51.64 (9.2)	51.99 (7.8)	59.78 (8.8)	**64.51** (6.9)
Win	4	7	5	-
Tie	1	0	1	-
Loss	5	3	4	-

SaDE samples random values for F and Cr from normal distributions $\mathcal{N}(0.5, 0.3)$ and $\mathcal{N}(CRm, 0.1)$, respectively. Initially, CRm is set to 0.5, but it is adapted every 25 generations (Qin et al., 2009). Similar adaptation occurs at every 50 generations to select a trial vector generation strategy (which can be “DE/rand/1/bin” or “DE/current-to-best/1/bin”). Although SaDE has a handful of preset parameters, it tends to be less sensitive to them in comparison to the original DE. This observation is also valid for the other DE variants. jDE, in particular, implements the strategy “DE/rand/1/bin” and adjusts F and Cr (at each generation), with probabilities $\tau_1 = \tau_2 = 0.1$, taking into account uniform distributions from $[0.1, 1]$ and $[0, 1]$, respectively. Initially, jDE assumes $F = 0.5$ and $Cr = 0.9$ (Brest et al., 2006). CoDE, by its turn, combines three different strategies, namely: “DE/rand/1/bin”, “DE/rand/2/bin”, and “DE/current-to-rand/1”, with three control parameter settings — $[F = 1.0, Cr = 0.1]$, $[F = 1.0, Cr = 0.9]$, and $[F = 0.8, Cr = 0.2]$ — in a random way to generate trial vectors. These strategies and control parameters

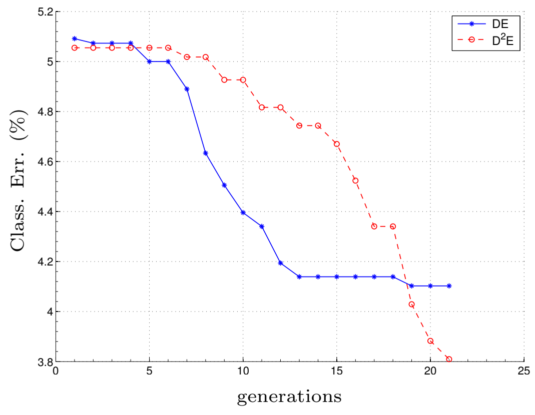

Figure 6: Average misclassification rates – *Wisconsin*.

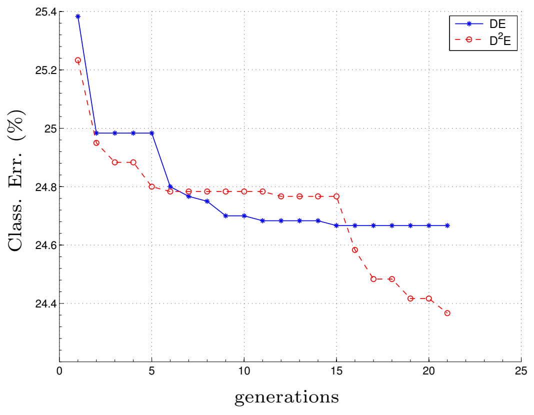

Figure 9: Average misclassification rates – *Blood*.

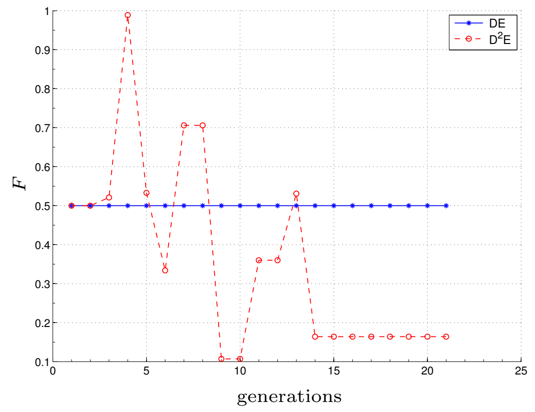

Figure 7: Values of F – *Wisconsin*.

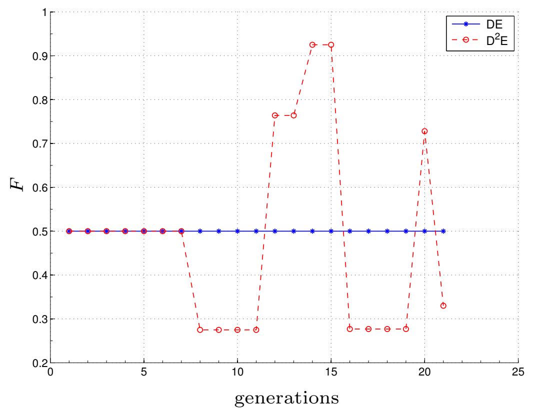

Figure 10: Values of F – *Blood*.

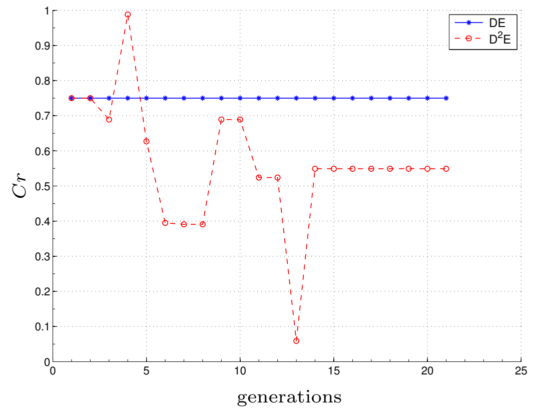

Figure 8: Values of Cr – *Wisconsin*.

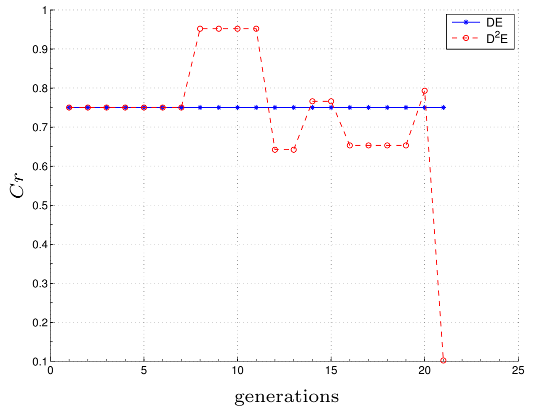

Figure 11: Values of Cr – *Blood*.

were preset following Wang et al. (2011). However, the authors stated that other settings can be used based on previous studies on the problem handled. From this viewpoint, DE variants can also be fine-tuned (by adjusting their preset parameters, if it is necessary). Since D^{2}E requires only the initial values of F and Cr, it has shown to be a user-friendlier alternative for optimising the C^{3}E-SL algorithm. Note that this advantage can also yield to computational savings in real-world applications.

5 Conclusions

We studied how to make an existing algorithm, named C^{3}E (from Consensus between Classification and Clustering Ensembles), more convenient by automatically tuning its main parameters — the relative importance of classifier and cluster ensembles (α) and the number of iterations of the algorithm (I) — with the use of an evolutionary algorithm named Dynamic Differential Evolution (D^{2}E). D^{2}E extends the Differential Evolution

(DE) algorithm by sampling values for its control parameters (F and Cr). Analyses of statistical significance conducted from experiments performed on ten datasets show that D^{2}E provides classifiers as accurate as those obtained from grid-search, but taking half the running time. This is particularly relevant for real-world data mining applications in which large datasets are available. D^{2}E also achieved, in most of the cases, equal or higher accuracies than DE across the datasets. Actually, D^{2}E holds the nice properties of DE, while offering robustness to initial parameter variations. From this viewpoint, D^{2}E is preferred over DE. In practice, for our application domain, D^{2}E is simple to implement and set up and has shown to be a good alternative to estimate parameters α and I of a simpler version of the C^{3}E algorithm based on a Squared Loss function (C^{3}E-SL). Furthermore, D^{2}E (with initial $F = Cr = 0.25$) showed competitive results in comparison with three state-of-the-art algorithms, namely SaDE, jDE, and CoDE. More specifically, we note that while D^{2}E provides classifiers as accurate as their counterparts, it is user-friendlier, for that it has shown to be robust to the initial choice of its parameters. Such advantage yields to computational savings by hopefully avoiding the task of fine tuning of the parameters, thus being relevant for real-world applications.

Finally, promising venues for future work involve using C^{3}E-SL with D^{2}E to solve difficult real-world problems in semi-supervised, active, and transfer-learning scenarios.

Acknowledgement: The authors would like to acknowledge Brazilian Research Agencies CNPq and FAPESP for the financial support. This research was also partially supported by ONR ATL Grant N00014-11-1-0105 and NSF Grants (IIS-0713142 and IIS-1016614).

References

Abdelaziz, A., Osama, R. A., and Elkhodary, S. M. (2012). Application of ant colony optimization and harmony search algorithms to reconfiguration of radial distribution networks with distributed generations. *Journal of Bioinformatics and Intelligent Control*, 1(1):86–94.

Acharya, A., Hruschka, E. R., Ghosh, J., and Acharyya, S. (2011). C3E: A framework for combining ensembles of classifiers and clusterers. In *Multiple Classifier Systems*, pages 269–278. LNCS Vol. 6713, Springer.

Ali, L. and Sabat, S. L. (2012). Particle swarm optimization based universal solver for global optimization. *Journal of Bioinformatics and Intelligent Control*, 1(1):95–105.

Bäck, T., Fogel, D. B., and Michalewicz, Z. (1997). *Handbook of Evolutionary Computation*. IOP Publishing, Bristol, UK, 1st edition.

Bäck, T., Hoffmeister, F., and Schwefel, H. P. (1991). A survey of evolution strategies. In *Proceedings of the Fourth International Conference on Genetic Algorithms*, pages 2–9.

Bäck, T. and Schwefel, H.-P. (1993). An overview of evolutionary algorithms for parameter optimization. *Evol. Comput.*, 1:1–23.

Banerjee, A., Merugu, S., Dhillon, I. S., and Ghosh, J. (2005). Clustering with bregman divergences. *Journal of Machine Learning Research*, 6:1705–1749.

Basu, S., Davidson, I., and Wagstaff, K. (2008). *Constrained Clustering: Advances in Algorithms, Theory, and Applications*. Chapman & Hall/CRC Press, 1 edition.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305.

Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. (2011). Distributed optimization and statistical learning via the alternating direction method of multipliers. *Found. Trends Mach. Learn.*, 3(1):1–122.

Brest, J., Greiner, S., Boskovic, B., Mernik, M., and Zumer, V. (2006). Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *Evolutionary Computation, IEEE Transactions on*, 10(6):646–657.

Campello, R. and Hruschka, E. (2006). A fuzzy extension of the silhouette width criterion for cluster analysis. *Fuzzy Sets and Systems*, 157(21):2858–2875.

Das, S. and Suganthan, P. (2011). Differential evolution: A survey of the state-of-the-art. *Evolutionary Computation, IEEE Transactions on*, 15(1):4–31.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30.

Eiben, A. E. and Smith, J. E. (2008). *Introduction to Evolutionary Computing (Natural Computing Series)*. Springer.

Fleming, P. and Purshouse, R. (2002). Evolutionary algorithms in control systems engineering: a survey. *Control Engineering Practice*, 10(11):1223 – 1241.

Frank, A. and Asuncion, A. (2010). UCI Machine Learning Repository [`http://archive.ics.uci.edu/ml`]. University of California, Irvine, School of Information and Computer Sciences.

Freitas, A. A. (2002). *Data Mining and Knowledge Discovery with Evolutionary Algorithms*. Springer-Verlag, New York, USA.

Freitas, A. A. (2003). *A survey of evolutionary algorithms for data mining and knowledge discovery*, pages 819–845. Springer-Verlag Inc., New York, USA.

Gamperle, R., Mller, S. D., and Koumoutsakos, P. (2002). A parameter study for differential evolution. In *WSEAS Int. Conf. on Advances in Intelligent Systems, Fuzzy Systems, Evolutionary Computation*, pages 293–298. Press.

Garcia-Gonzalo, E. and Fernandez-Martinez, J. (2012). A brief historical review of particle swarm optimization (PSO). *Journal of Bioinformatics and Intelligent Control*, 1(1):3–16.

Ghosh, J. and Acharya, A. (2011). Cluster ensembles. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, 1(4):305–315.

Hruschka, E. R., Campello, R. J., and Castro, L. N. D. (2006). Evolving clusters in gene-expression data. *Information Sciences*, 176(13):1898–1927.

Jain, A. K. (2010). Data clustering: 50 years beyond k-means. *Pattern Recognition Letters*, 31(8):651 – 666.

Kantardzic, M. (2011). *Data mining: concepts, models, methods, and algorithms*. John Wiley & Sons.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942 –1948.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science, Number 4598*, 220, 4598:671–680.

Kittler, J. and Roli, F. (2002). *Multiple Classifier Systems*. Springer.

Kuncheva, L. I. (2004). *Combining Pattern Classifiers: Methods and Algorithms*. Wiley-Interscience, Hoboken, NJ.

Liu, J. and Lampinen, J. (2002). On setting the control parameter of the differential evolution method. In *Proc. 8th int. conf. soft computing (MENDEL 2002)*, pages 11–18.

Madden, S. (2012). From databases to big data. *IEEE Internet Computing*, 16(3).

Natarajan, A., Subramanian, S., and Premalatha, K. (2012). A comparative study of cuckoo search and bat algorithm for bloom filter optimisation in spam filtering. *International Journal of Bio-Inspired Computation*, 4(2):89–99.

Oza, N. C. and Tumer, K. (2008). Classifier ensembles: Select real-world applications. *Information Fusion*, 9(1):4–20.

Price, K. (1996). Differential evolution: a fast and simple numerical optimizer. In *Fuzzy Information Processing Society, 1996. NAFIPS. 1996 Biennial Conference of the North American*, pages 524–527. IEEE.

Price, K., Storn, R. M., and Lampinen, J. A. (2005). *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Qin, A. K., Huang, V. L., and Suganthan, P. (2009). Differential evolution algorithm with strategy adaptation for global numerical optimization. *Evolutionary Computation, IEEE Transactions on*, 13(2):398–417.

Ronkkonen, J., Kukkonen, S., and Price, K. V. (2005). Real-parameter optimization with differential evolution. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 1, pages 506–513. IEEE.

Storn, R. (1996). On the usage of differential evolution for function optimization. In *Fuzzy Information Processing Society, 1996. NAFIPS., 1996 Biennial Conference of the North American*, pages 519–523.

Storn, R. and Price, K. (1997). Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Jornal of Global Optimization*, 11(4):341–359.

Strehl, A. and Ghosh, J. (2002). Cluster ensembles - a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617.

Tumer, K. and Ghosh, J. (1996). Analysis of decision boundaries in linearly combined neural classifiers. *Pattern Recognition*, 29(2):341–348.

Wang, Y., Cai, Z., and Zhang, Q. (2011). Differential evolution with composite trial vector generation strategies and control parameters. *Evolutionary Computation, IEEE Transactions on*, 15(1):55–66.

Witten, I. H. and Frank, E. (2005). *Data mining: practical machine learning tools and techniques*. Morgan Kaufmann Series in Data Management Systems. 2nd edition.

Yang, X.-S. (2010). *Nature-Inspired Metaheuristic Algorithms*. Luniver Press, 2nd edition.

Yang, X.-S. and Deb, S. (2012). Two–stage eagle strategy with differential evolution. *International Journal of Bio-Inspired Computation*, 4(1):1–5.

Yu, B., Cui, Z., and Zhang, G. (2013). Artificial plant optimization algorithm with correlation branches. *Journal of Bioinformatics and Intelligent Control*, 2(2):146–155.

Zhong, Y., Wang, L., Wang, C., and Zhang, H. (2012). Multi–agent simulated annealing algorithm based on differential evolution algorithm. *International Journal of Bio-Inspired Computation*, 4(4):217–228.