1. Introduction

This work introduces a stochastically optimized, granular k-nearest neighbor (knn) classifier based on INs, where IN stands for Intervals’ Number. An IN is a mathematical object, which may represent either a fuzzy interval or a probability distribution (Papadakis and Kaburlasos, 2010). In any case, an IN can be interpreted as an information granule.

INs have been studied in a series of publications. In particular, as explained in Kaburlasos et al. (2013a), it has been shown that the set \( \mathbb{F}_1 \) of INs is a metric lattice with cardinality \( \aleph_1 \), where \( \aleph_1 \) is the cardinality of the set \( \mathbb{R} \) of real numbers; moreover, \( \mathbb{F}_1 \) is a cone in a linear space. Note that previous work (Kaburlasos, 2004; Kaburlasos, 2006) has employed the term FIN (i.e., fuzzy interval number) instead of the term IN because it stressed a fuzzy interpretation. Likewise, the term CALFIN, proposed previously for an algorithm which induces a FIN from a population of measurements, was later replaced by the term CALCIN (Papadakis and Kaburlasos, 2010). Recall that an IN computed by algorithm CALCIN retains all-order data statistics (Kaburlasos et al., 2013a). In the aforementioned context, the capacity as well as the rich potential of INs, especially in industrial applications, has been demonstrated (Kaburlasos and Kehagias, 2014; Kaburlasos and Pachidis, 2014; Kaburlasos and Pachidis, 2014).

INs have been used in an array of computational intelligence applications regarding clustering, classification and regression (Kaburlasos and Moussiades, 2014; Kaburlasos and Pachidis, 2014; Kaburlasos and Papadakis, 2006; Kaburlasos et al., 2012, 2013a; Papadakis and Kaburlasos, 2010; Papadakis et al., 2014). There is experimental evidence that a parametric, IN-based scheme can be optimized toward clearly improving performance. More specifically, optimization has been pursued by stochastic search techniques including genetic algorithms (Kaburlasos and Papadakis, 2006; Kaburlasos et al., 2012, 2013a; Papadakis and Kaburlasos, 2010) and particle swarm optimization (Papadakis et al., 2014) since, currently, there are no analytic optimization methods available in the lattice of INs.

Previous works have frequently employed an inclusion measure (\( \sigma \)) function as an instrument for decision making in the lattice of INs (Kaburlasos and Pachidis, 2014; Kaburlasos et al., 2012, 2013a; Papadakis et al., 2014). The interest of this work is in (fuzzy) nearest neighbor classification (Derrac et al., 2014). Note that, lately, a number of knn classifiers based on INs, namely INknn classifiers, have been introduced (Kaburlasos et al., 2014; Pachidis and Kaburlasos, 2012; Tsoukalas et al., 2013); nevertheless, none of the latter classifiers was optimized. On the grounds of compelling evidence, as explained above, this work proposes an optimized INknn classifier toward improving performance. We remark that various heuristic optimization methods have been proposed in machine learning including Simulated Annealing (SA) (Chang et al., 1994), Ant Colony (Chi and Yang, 2006), Particle Swarm Optimization (PSO) algorithms (Huang and Dun, 2008; Lotfi Shahreza et al., 2011), Differential Evolution (DE) (Liu and Sun, 2011; Si et al., 2012), Genetic Algorithms (GAs) (Polat and Yildirim, 2008), etc. Lately, Rashedi and colleagues have proposed the
gravitational search algorithm (GSA) [Rashedi et al., 2009], that is a swarm based meta-heuristic search algorithm based on the Newtonian laws of gravity. The GSA has already been successfully applied to numerous problems [Bahroloulool et al., 2012; Li and Zhou, 2011; Rashedi et al., 2011; Rebollo-Ruiz and Graña, 2012, 2013; Rebollo et al., 2012; Sarafrazi and Nezamabadi-pour, 2013; Taghipour et al., 2010; Yin et al., 2011]. This work proposes a synergy of GSA with the INKnn classifier toward improving the capacity of the latter classifier. Hence, the (granular) gsaINknn classifier emerges whose capacity is demonstrated here comparatively on 12 benchmark datasets.

In a more general context, the proposed gsaINknn classifier is a scheme of the emerging lattice computing (LC) paradigm. Note that LC was originally defined as “the collection of Computational Intelligence tools and techniques that either make use of lattice operators inf and sup for the construction of the computational algorithms or exploit lattice theory for language representation and reasoning” (Graña, 2009). Recent work has extended the meaning of LC as “an evolving collection of tools and methodologies that process lattice ordered data per se including logic values, numbers, sets, symbols, graphs, etc.” (Kaburlasos and Kaburlasos, 2014; Kaburlasos et al., 2013a, 2013b). The LC paradigm provides instruments for granular computing, where uncertainty/ambiguity is accommodated in partially/lattice-ordered information granules (Jamshidi and Nezamabadi-pour, 2013; Kaburlasos, 2010; Liu et al., 2013; Sussner and Esmi, 2011).

A number of LC models have already been proposed in the context of mathematical morphology. For instance, morphological neural networks (MNN) including both morphological perceptions and fuzzy morphological associative memories (FMAMs) (Sussner and Esmi, 2009, 2011; Sussner and Vallee, 2006; Vallee and Grande Vicente, 2012) can be classified as LC models. In particular, Sussner and colleagues have employed a FMAM to implement a fuzzy inference system based on the complete lattice structure of the class of fuzzy sets (Sussner and Vallee, 2006; Vallee and Sussner, 2008, 2011). Furthermore, Graña and colleagues have applied LC techniques to image analysis applications of mathematical morphology (Graña et al., 2011, 2010, 2009). Of particular interest in LC is the notion of a fuzzy lattice, which has been proposed by Nanda toward fuzzifying a partial order relation (Nanda, 1989). Working independently Kaburlasos and colleagues, inspired from the adaptive resonance theory (ART) for neural computation (Carpenter et al., 1991, 1992), have proposed a number of fuzzy lattice neural networks for clustering and classification (Kaburlasos, 2006) operating on fuzzy lattice reasoning (FLR) principles. The FLR classifier was introduced in Kaburlasos et al. (2007) for inducing descriptive decision-making knowledge (rules) in a mathematical lattice data domain, including the space $\mathbb{R}^d$ as a special case; moreover, the FLR classifier has been successfully applied to a variety of problems such as ambient ozone estimation as well as air quality assessment (Athanasiadis and Kaburlasos, 2006). Recent trends in lattice computing appear in Graña (2012), Kaburlasos (2011), Kaburlasos and Ritter (2007).

The layout of this paper is as follows. Section 2 outlines the mathematical background. Section 3 presents the INKnn classifier including an explanatory application example. Section 4 describes the GSA optimization algorithm. Section 5 details the gsaINknn classifier. Section 6 presents comparatively experimental results regarding 12 benchmark classification datasets. Finally, Section 7 concludes by both summarizing our contribution and delineating future work.

## 2. Mathematical background

This section outlines general lattice notions followed by a hierarchy of lattices ending up to Intervals’ Numbers, or INS for short.

### 2.1. General lattices

A set $P$ with a partial order ($\sqsubseteq$) relation $\sqsubseteq$ is called partially ordered set or poset for short, symbolically $(P, \sqsubseteq)$ (Birkhoff, 1967; Kaburlasos, 2006; Kaburlasos and Kaburlasos, 2014; Kaburlasos and Pachidis, 2014; Rutherford, 1965). A function $\varphi: P \rightarrow \{0, 1\}$ from a poset $(P, \sqsubseteq)$ to a poset $(Q, \sqsubseteq)$ is called isomorphic if $\varphi(x) \sqsubseteq \varphi(y)$ if and only if $x \sqsubseteq y$. It is well known that the inverse $\varphi^{-1}$, namely dual ($\sqsubseteq$), of an order relation $\sqsubseteq$ is itself an order relation. A lattice $(L, \sqsubseteq)$ is a poset with the additional property that any two of its elements $a, b \in L$ have both an infimum denoted by $\inf(a, b)$ and a supremum denoted by $\sup(a, b)$. The lattice operations $\land$ and $\lor$ are called meet and join, respectively. A lattice $(L, \sqsubseteq)$ is called totally-ordered if for $a, b \in L$ it is either $a \sqsubseteq b$ or $b \sqsubseteq a$. In this work we use “square symbols” such as $\land$, $\lor$ and $\sqsubseteq$ with general lattice elements, “straight symbols” $\land$, $\lor$ and $\sqsubseteq$ with real numbers, and symbols $\land$, $\lor$ and $\sqsubseteq$ with sets.

An aggregate lattice $(L, \sqsubseteq)$ is the Cartesian product of $N$ component lattices $L_1, \ldots, L_n$; i.e. $(L, \sqsubseteq)=(L_1 \times \cdots \times L_n, \sqsubseteq)$. The product lattice $L$ operates on fuzzy lattice $(L, \sqsubseteq)$ that is defined as

\[ (a_1, \ldots, a_n)(b_1, \ldots, b_n) = (a_1 \land b_1, \ldots, a_n \land b_n) \quad \text{and} \quad (a_1, \ldots, a_n)(\bot) = (a_1 \land \bot, \ldots, a_n \land \bot). \]

A valuation on a lattice $(L, \sqsubseteq)$ is a real function $\nu: L \rightarrow \mathbb{R}$ which satisfies $\nu(a) + \nu(b) = \nu(a \lor b) + \nu(a, b)$ and $\nu(a) < \nu(b)$. A positive valuation function $\nu: L \rightarrow \mathbb{R}$ implies a metric function $d: L \times L \rightarrow \mathbb{R}$ given by $d(x, y) = \nu(x \land y) - \nu(x) - \nu(y)$. A metric distance function $d$ is called a generalized interval $(\Delta, \sqsubseteq)$ that is defined as $\Delta = \{(a, b) \mid a \sqsubseteq b \}$.

When each of its subsets has a supremum as well as an infimum in $L$. A non-void complete lattice has both a least element and a greatest element denoted by $\bot$ and $\top$, respectively. A lattice $(L, \sqsubseteq)$ is called upper-bounded if for a $b \in L$ it is either $a \sqsubseteq b$ or $b \sqsubseteq a$. Any strictly increasing real function $\theta: L \rightarrow \mathbb{R}$ provides instruments for granular computing, where uncertainty/ambiguity is accommodated in partially/lattice-ordered information granules (Jamshidi and Nezamabadi-pour, 2013; Kaburlasos, 2010; Liu et al., 2013; Sussner and Esmi, 2011).

A valuation on a lattice $(L, \sqsubseteq)$ is a real function $\nu: L \rightarrow \mathbb{R}$ which satisfies $\nu(a) + \nu(b) = \nu(a \lor b) + \nu(a, b)$ and $\nu(a) < \nu(b)$. A positive valuation function $\nu: L \rightarrow \mathbb{R}$ implies a metric function $d: L \times L \rightarrow \mathbb{R}$ given by $d(x, y) = \nu(x \land y) - \nu(x) - \nu(y)$. A metric distance function $d$ is called an isomorphic function $\theta: L \rightarrow L$ that is defined as $\theta(a, b) = \nu(\theta(a)) + \nu(b)$.

When each of its subsets has a supremum as well as an infimum in $L$. A non-void complete lattice has both a least element and a greatest element denoted by $\bot$ and $\top$, respectively. A lattice $(L, \sqsubseteq)$ is called upper-bounded $(\Delta, \sqsubseteq)$ that is defined as $\Delta = \{(a, b) \mid a \sqsubseteq b \}$. A generalized interval $(\Delta, \sqsubseteq)$ is a real function $\nu: L \rightarrow \mathbb{R}$ which satisfies $\nu(a) + \nu(b) = \nu(a \lor b) + \nu(a, b)$ and $\nu(a) < \nu(b)$.

A metric distance function $d$ is defined on $(\Delta, \sqsubseteq)$ as follows:

\[
d_{\Delta}(a, b) = \nu(A)(a, b) = \nu_{\Delta}(a, b) = \nu_{\Delta}(a, b) = \nu_{\Delta}(a, b) = \nu_{\Delta}(a, b) = \nu_{\Delta}(a, b) = \nu_{\Delta}(a, b).
\]

We define the set of conventional intervals as $\mathbb{I}(L) = \{(a, b) \mid a, b \in L \land a \sqsubseteq b\}$. Augmenting $\mathbb{I}(L)$ by the empty interval, denoted by $\emptyset$,
there follows the complete lattice \((j_1 = \tilde{j}(L) = \tilde{j}(L) \cup (I \setminus j(I)), \leq)\), namely lattice of Type-1 intervals, or simply intervals, ordered by the set inclusion relation \((\subseteq)\). The least and greatest element in lattice \((j_1, \leq)\) are denoted by \(\emptyset = [0,0]\) and \(I = [0,1] = L\), respectively.

Consider the following definition.

**Definition 1.** A Type-1 Intervals: Number, or (Type-1) IN for short, is a (Type-1) interval function \(F: [0,1] \rightarrow \mathbb{J}_1\) which satisfies (1) \(h_1 \leq h_2 \Rightarrow F_{h_1} \supseteq F_{h_2}\) and (2) \(\forall P \subseteq [0,1]; \cap_{h \in P} F_h = F_{\cap P}\).

Let \(F_i\) denote the set of INs. It turns out that \((F_1, \leq)\) is a complete lattice with order \(F_i \leq G_j \iff (\forall h \in [0,1]: F_i \subseteq G_j) \iff (\forall x \in L: F(x) \leq G(x))\). We remark that there are two equivalent representations for an IN \(F_i\), namely the interval-representation \(F_{\leq} : [0,1] \rightarrow \mathbb{J}_1\) and the membership-function-representation \(F_{\leq} : R \rightarrow [0,1]\).

An IN \(F\) is called trivial iff \(F_i = [0,1]\), or \(F_i = [a,0]\) \(\forall h \in [0,1]\) and \(a \in L\). The following proposition introduces a metric on lattice \((F_1, \leq)\).

**Proposition 1.** Let \(F, G\) be INs in lattice \((F_1, \leq)\). Assuming that the following integral exists, a metric function \(d_{F_i}(\cdot, \cdot)\) is given by \(d_{F_i}(F,G) = \int_{a}^{b} d_{F_i}(F_h,G_h) dh\).

Given two INs \(F = (F_1, ..., F_n) \in \mathbb{F}^n\) and \(G = (G_1, ..., G_n) \in \mathbb{F}^n\), a Minkowski metric \(d_{\mathbb{F}} : \mathbb{F}^n \times \mathbb{F}^n \rightarrow [0,1]\) is defined as \(d_{\mathbb{F}}(F,G) = \langle d_{F_1}(F_1, G_1) \rangle^p + \ldots + \langle d_{F_n}(F_n, G_n) \rangle^p \rangle^{1/p}\).

A Type-2 interval is defined as an interval of Type-1 interval; moreover, a Type-2 IN is defined as an interval of Type-1 INs (Kaburlasos et al. 2012, 2013a). It turns out that the aforementioned Minkowski metric can extend to N-tuple Type-2 INs.

An IN can be induced from a set \(\{x_1, ..., x_n\}\) of data samples by defining a piecewise linear, strictly increasing, cumulative real function \(c: R \rightarrow [0,1]\) such that \(c(x_i) = 1/|\{x_j: j \in \{1, ..., n\} \text{ and } x_i \leq x_j\}|, i \in \{1, ..., n\}\); where \(|S|\) denotes the cardinality of the set \(S\). Let \(x_0\) be such that \(c(x_0) = 0.5\). Finally, \(F_i\) is defined such that for \(x \leq x_0\) it is \(F_i(x) = 2c(x)\), whereas for \(x > x_0\) it is \(F_i(x) = 2(1 - c(x))\) as illustrated in Kaburlasos et al. (2012) (Section 4.1). In practice, the membership function \(F_1: L \rightarrow [0,1]\) is specified by two vectors \(pts\) and \(val\) which hold the corresponding abscissa values and ordinate values, respectively, of the membership function \(F_1\). In the context of this work, we define the weight of an IN as the number of data samples used to induce IN \(F\).

### 3. The INknn classifier

In the following we present an extension, namely INknn, of the conventional knn classifier for \(k = 1\) in the metric lattice \((F_1, \leq)^n\) of INs (see in **The INknn algorithm**).

In words, the INknn classifier operates by calculating the (metric) distance of an unlabeled testing datum \(F_0 = F_1^n\) from all the labeled training data. In conclusion, \(F_0\) is assigned the label of its nearest training datum \(F_i\) provided that the corresponding distance is less than a user-defined threshold \(\mu_0\). Otherwise, if \(d_2(F_0, F_i) > \mu_0\), then \(F_0\) is of “unknown” class. Note that the proposed INknn classifier uses the Minkowski metric \(d_2: \mathbb{F}^n \times \mathbb{F}^n \rightarrow [0,1]\).

The following example illustrates the relation between IN weight (the latter was defined at the very end of the previous section) and Classification Accuracy; the latter is the percentage of correctly classified data and it can be computed by Eq. (8).

### The INknn Algorithm

1. Consider labeled, training data \((F_1, c_1), ..., (F_n, c_n)\), where \(F_i \in \mathbb{F}^n\), \(c_i \in L\) for \(i \in \{1, ..., n\}\) and \(L\) is a set of class labels.
2. Consider a new, unlabeled (testing) datum \(F_0 \in \mathbb{F}^n\) for classification.
3. For each training datum \(F_i, i \in \{1, ..., n\}\) compute the Minkowski distance \(d_2(F_0, F_i)\).
4. Let \(j = \arg \min_{i \in \{1, ..., n\}} \{d_2(F_0, F_i)\}\).
5. The class label of \(F_0\) is defined to be \(c_j \in L\) given that \(d_2(F_0, F_{j}) \leq \mu_0\), where \(\mu_0\) is a user-defined threshold; otherwise, if \(d_2(F_0, F_{j}) > \mu_0\), then the class of \(F_0\) is “unknown”.

### An Application Example

The spiral problem involves a non-trivial benchmark dataset for testing the capacity of a learning scheme (Carpenter et al. 1992; Lang and Witbrock, 1988). The corresponding dataset includes sample points \((x_0, y_0)\) from two 2D spirals given by \((x_0, y_0) = (r_0 \sin o_0 + 0.5, r_0 \cos o_0 + 0.5)\), where \(r_0 = 0.4(105 - n)/104, o_0 = \pi(n - 1)/16\). In particular, here we employed sample points \((x_0, y_0)\), \(n = 1, ..., 97\) from the first spiral class as well as sample points \((1 - x_0, 1 - y_0)\), \(n = 1, ..., 97\) from the second spiral class shown in Fig. 1(a); hence, we employed a total of \(97 + 97 = 194\) sample points or, equivalently, 194 data.

We carried out a series of INknn classification experiments such that, after shuffling randomly the 194 data before an experiment, we engaged 10,20, ..., 170 and 180 data, for training. In particular, half of the engaged data originated from each spiral class. In conclusion, in each experiment we used all the training data available from a class in order to compute a single (2-tuple) IN per class. Hence, two (2-tuple) INs were calculated per experiment with (sum of) weights equal to 10,20, ..., 170 and 180. The previous complete step 1 of the INknn algorithm. In the remaining steps of the INknn algorithm, a testing datum \((x_0, y_0)\) was represented by a pair \((x_0, y_0, [y_0, y_0])\) of trivial INs for \(h \in [0,1]\).

**Fig. 1.** (a) The two-classes Spiral dataset and (b) Training/testing classification accuracy versus IN weight.
Fig. 1(b) displays the Classification Accuracy regarding the training/testing datasets versus the (sum of) IN weights per experiment. In particular, Fig. 1(b) shows that accommodating ever more data into the two INs, first, typically keeps the Classification Accuracy on the training dataset in the range 50–60% due to the intermingling of the two classes and, second, it progressively increases Classification Accuracy on the testing dataset due to an improvement of INknn’s capacity for generalization especially as the number of the testing data decreases.

The optimal number of INs per class depends on the data. For instance, were all the data of a class both near each other and apart from the data in any other class then one (N-tuple) IN per class would suffice; otherwise, more than one INs are required per class. The optimal number of INs in a specific application is not known “a priori” but rather it is to be induced from the data.

4. The gravitational search algorithm

This section delineates the Gravitational Search Algorithm, or GSA for short, that is a population-based stochastic search heuristic (Rashedi et al., 2009, 2011), where a population member is called agent. An agent represents (part of) the training data with k (N-tuple) INs per class in N data dimensions. The mechanics of GSA is inspired from the Newtonian laws of gravity and motion. More specifically, a GSA agent is dealt with as a physical object with mass. The position of an agent represents a solution to an optimization problem, whereas an agent’s mass represents the corresponding Classification Accuracy such that larger masses correspond to better solutions. All agents interact with one another with gravity forces resulting in motion. During the lifetime of GSA, agents with large masses grow larger and move more slowly toward computing better solutions. The basic equations of the GSA are presented next.

Given a user-defined number Na of agents, we define the position Xk of agent i as

\[ X_k = (x_{k1}, ..., x_{kn}), \quad i = 1, ..., N_a, \]

where N is the search space dimension. A normalized mass for agent i is computed as follows:

\[ M_i(t) = \frac{f_{fit_i}(t) - \text{worst}(t)}{\sum_{i=1}^{N_a} (f_{fit_i}(t) - \text{worst}(t))}, \quad i = 1, ..., N_a, \]

where \( f_{fit_i}(t) \) is the fitness value (calculated by Eq. (8)) of agent i at time \( t \in \{1, ..., T\} \). worst(\( t \)) for a maximization (resp. minimization) optimization problem is the minimum (resp. maximum) fitness of all agents.

We compute the next position \( x_k(t+1) \) of agent \( i \in \{1, ..., N_a\} \) in dimension d at time \( t+1 \) as follows. First, we use Eq. (4) to compute the force \( F_k^d(t) \) at time t applied by the K largest masses in the set “Kbest”; second, we compute the corresponding acceleration \( a_k^d(t) \) by Eq. (5); third, we compute the velocity \( v_k^d(t+1) \) at time \( t+1 \) as a fraction of the velocity at time t increased by the acceleration at time t according to Eq. (6); finally, we compute \( x_k^d(t+1) \) using Eq. (7). 

\[ F_k^d(t) = \sum_{j \in \text{Kbest}_i} (\text{rand}_j G(t) M_j(t) R_k^d(t) + \varepsilon (x_j^d(t) - x_k^d(t))) \]

(4)

\[ a_k^d(t) = \frac{F_k^d(t)}{M_k(t)} = \sum_{j \in \text{Kbest}_i, j \neq i} (\text{rand}_j G(t) M_j(t) R_k^d(t) + \varepsilon (x_j^d(t) - x_k^d(t))) \]

(5)

\[ v_k^d(t+1) = (\text{rand}_i v_k^d(t)) + a_k^d(t) \]

(6)

\[ x_k^d(t+1) = x_k^d(t) + v_k^d(t+1) \]

(7)

where

- both “\( \text{rand}_j \)” and “\( \text{rand}_i \)” are random numbers, uniformly distributed over the interval [0,1],
- “\( R_k^d(t) \)” is the Euclidean distance between agents i and j,
- “\( \varepsilon \)” is a small, positive number (to avoid dividing by zero),
- “Kbest” is the set of the K largest masses, where K is a decreasing function of time t with a user-defined initial value \( K(t=0)=K_0=N_a \) and final value \( K(t=T)=1 \), and
- “\( G(t) = C_0 e^{-\alpha t/t_0} \)” is the gravitational constant with initial value \( C_0 \), \( \alpha \) is a user-defined constant, \( t \in \{1, ..., T\} \) is the iteration number, and \( T \) is the total number of iterations.

Note that Eq. (4) uses “\( R_k^d(t) \)” instead of the square “\( R_k^d(t)^2 \)” in Newton’s laws because it was confirmed empirically that “\( R_k^d(t) \)” typically produces better results than “\( R_k^d(t)^2 \)”.

5. The gsaINknn classifier

Given a dataset for training in the N-dimensional Euclidean space with a number of C classes, the objective is to compute an optimal INknn classifier using GSA heuristics.

Using a random training dataset partition (including \( N_a \) parts) we assume an initial population of \( N_a \) agents such that a different agent corresponds to a different partition part. In conclusion, an agent is represented by a \( (C \times k \times N \times S) \)-dimensional vector, where C is the number of classes, k denotes the (user-defined) maximum number of INs per class, N is the dimension of the data (feature) space, moreover S equals the (user-defined) number of abscissa/ordinate samples used to represent an IN membership function. The objective is to compute an agent that produces as large classification accuracy as possible. Note that, during training, only the \( S \times L \) abscissa (interval ends) values of an IN membership function update, whereas the corresponding L ordinate values remain constant spaced uniformly from 0 to 1 included.

The fitness function, for computing the mass \( M_i(t) \) of agent \( i \in \{1, ..., N_a\} \) at iteration time \( t \in \{1, ..., T\} \), was calculated as follows.

\[ \text{fit}_i = \frac{CC}{CC + IC} \times 100 \]

(8)

where CC and IC denote the number of correctly and incorrectly classified data, respectively. The objective is to maximize \( \text{fit}_i \) for an agent \( i \in \{1, ..., N_a\} \). A cycle of computing the next agent position repeats until the stop condition \( t > T \) is met as shown in The gsaINknn classifier algorithm. Recall the final value “\( K(t=T) = 1 \)” mentioned in the previous section; that is, only a single agent remains in the set Kbest at the termination of the gsaINknn classifier algorithm. Since there is no guarantee that the single agent at time \( t=T \) corresponds to the best solution, we keep storing the best solution calculated during the lifetime of the gsaINknn classifier algorithm.

The gsaINknn classifier algorithm

For a classification problem with a number of C classes in the N-dimensional feature space:

1. A user-defines: The number \( N_a \) of agents in the population; the maximum number \( K \) of INs per class; the initial number \( K(t=0)=K_0=N_a \) of largest masses; the total number \( T \) of iterations; the number \( S \) of abscissa values of an IN membership function;
2. Randomly initialize a \( (C \times k \times N \times S) \)-dimensional swarm of \( N_a \) agents;
3. while \( t \leq T \) do
4. Decrease the integer number \( K \); increase the time \( t \) by 1;
5. Identify the set Kbest of the largest K masses;
6. for each agent \( i = 1, ..., N_a \) do
7. Evaluate the fitness function using Eq. (8);
8. Calculate the normalized mass using Eq. (3);


9. Calculate the acceleration using Eq. (5);
10. Update the velocity using Eq. (6);
11. Update the next position using Eq. (7);
12. end for
13. end while
14. Best solution found.

6. Experiments and results

This section demonstrates the application of the gsaINknn classifier comparatively with alternative classifiers from the literature on 11 benchmark datasets from the University of California Irvine repository of machine learning databases (Frank and Asuncion, 2010). In addition, we used Ripley’s benchmark dataset from Web site http://www.stats.ox.ac.uk/pub/PRNN/.

6.1. Benchmark datasets

Table 1 displays selected characteristics of the 12 benchmark datasets used in this work. In particular, note that the original Ecoli benchmark dataset includes 336 instances partitioned in 8 classes. Nevertheless, since 3 of the classes contain only 2, 2 and 5 instances, we decided to omit them. Hence, we employed a modified Ecoli dataset with 5 classes including 327 instances. Furthermore, since one of the 10 classes of the original Yeast dataset includes only 5 instances, we decided to omit it. In conclusion, we employed a modified Yeast benchmark dataset with 9 classes including 1479 instances.

6.2. Data preprocessing and classifier initialization

In a data preprocessing step, N-dimensional instances, where N is the corresponding number of attributes/dimensions, were normalized in the unit hypercube [0,1] by replacing an attribute value x by the normalized value \( x_{\text{norm}} = (x - x_{\text{min}}) / (x_{\text{max}} - x_{\text{min}}) \), where \( x_{\text{min}} \) and \( x_{\text{max}} \) stand for the corresponding minimum and maximum attribute values, respectively.

We employed a number of alternative classifiers from the literature including the GSA (Bahrololoum et al., 2012), fuzzy-ART (Carpenter et al., 1991), (conventional) knn (Cunningham and Delany, 2007), SOM (Kohonen and Somervuo, 2002), Inknn (Pachidis and Kaburlasos, 2012), GRNN (Specht, 1991) and Support Vector Machine (SVM) (Vapnik, 1999) all implemented in the C++ programming language. For a fair comparison we used identical datasets for training/testing. When a training/testing dataset was not given explicitly for a benchmark dataset, we engaged a random 70% of the instances for training; whereas the remaining instances were used for testing. Care was taken so that each class is represented fairly in the dataset for training as well as in the dataset for testing; that is, each class was represented in the training/testing dataset in proportion to its corresponding total number of instances in the dataset. A missing value in an instance attribute was replaced by the corresponding attribute average. The initialization of different classifiers is described next.

Computational experiments with the SOM classifier were carried out using a 4 x 4 grid of units for 100 epochs. Since the results by SOM depend on the initial values of the weights, we chose the weights that yielded the best results on the training dataset in 10 random initializations. The GRNN classifier was considered with values of the variance parameter between 0 and 0.5 in steps of 0.001. For the fuzzy-ART we assumed a choice parameter value equal to 0.01; furthermore, the values of the vigilance parameter were between 0 and 1 in steps of 0.01. For both knn and Inknn classifiers, k=1 was user-defined; that is, only the (one) nearest neighbor was considered. Regarding the SVM classifier, we employed the “one versus all” binary classification method; moreover, for the SVM classifier here we employed Gaussian radial basis function kernels with a default scaling factor sigma of 1; furthermore, we used the Sequential Minimal Optimization (SMO) method to compute the separating hyperplanes of the SVM classifier. The parameters for the GSA classifier were tuned as described in Bahrololoum et al. (2012). More specifically, a population of \( N_{s} \) agents was user-defined so as to cover the training data. Moreover, the gsaINknn classifier ran for \( T=100 \) iterations with an initial value of the \( G \) parameter in the range [0.5,5.0]. The value of “k” in the product “C x k x N x S” was (user-defined to) \( k=1 \). In addition, “S” was user-defined to \( S=2L=2 \times 32=64 \). We point out that the number of data employed for inducing an IN by either classifier INknn or gsaINknn depended on the specific (benchmark) classification dataset. In addition, the functions \( \theta(x)=1-x \) and \( \varphi(x)=x \) were employed by both classifiers INknn and gsaINknn.

6.3. Comparative experimental results and discussion

Tables 2 and 3 display (a) the average classification accuracy in 10 computational experiments for different (random) data permutations, and (b) the ranking (within parentheses) of each classifier for each benchmark dataset regarding training and testing, respectively. In other words, each table cell displays the average (percentage) correct classification of a specific classifier on a specific dataset in 10 computational experiments. The best result for each benchmark dataset is displayed in bold. Note that the training data classification accuracy of the knn classifier on the Haberman’s Survival benchmark (Table 2) is less than 100% because some training data are erroneously repeated with different class labels.

---

Table 1
Selected characteristics of the 12 benchmark datasets used in this work.

<table>
<thead>
<tr>
<th>Benchmark dataset name</th>
<th>No. of instances</th>
<th>Missing value</th>
<th>No. of training data</th>
<th>No. of testing data</th>
<th>No. of attributes</th>
<th>No. of classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Dermatology</td>
<td>366</td>
<td>Yes</td>
<td>258</td>
<td>108</td>
<td>34</td>
<td>6</td>
</tr>
<tr>
<td>2. Ecoli</td>
<td>327</td>
<td>No</td>
<td>237</td>
<td>90</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>3. Haberman’s survival</td>
<td>306</td>
<td>No</td>
<td>216</td>
<td>90</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4. Iris</td>
<td>150</td>
<td>No</td>
<td>105</td>
<td>45</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>5. Pima Indians diabetes</td>
<td>768</td>
<td>No</td>
<td>543</td>
<td>225</td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td>6. Ripley’s</td>
<td>1250</td>
<td>No</td>
<td>250</td>
<td>1000</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>7. Wine</td>
<td>178</td>
<td>No</td>
<td>130</td>
<td>48</td>
<td>13</td>
<td>3</td>
</tr>
<tr>
<td>8. Credit approval</td>
<td>690</td>
<td>Yes</td>
<td>483</td>
<td>207</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>9. Statlog (heart)</td>
<td>270</td>
<td>No</td>
<td>189</td>
<td>81</td>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>10. Thyroid</td>
<td>215</td>
<td>No</td>
<td>155</td>
<td>60</td>
<td>5</td>
<td>3</td>
</tr>
<tr>
<td>11. Yeast</td>
<td>1479</td>
<td>No</td>
<td>1038</td>
<td>441</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>12. Breast Cancer Wisconsin</td>
<td>569</td>
<td>No</td>
<td>401</td>
<td>168</td>
<td>30</td>
<td>2</td>
</tr>
</tbody>
</table>
Both Tables 2 and 3 demonstrate the well-known fact that there is no “universally optimal classifier”, in the sense that a classifier may perform well on some datasets and poorly on other ones (De Falco et al., 2007). On the one hand, Table 2 demonstrates the very good accuracy of both GRNN and kNN classifiers: their ranking is either 1 or 2. Nevertheless, both GRNN and kNN need to store the entire training dataset. Both classifiers SOM and fuzzy-ART have demonstrated a moderate performance. The INkNN classifier typically achieved one of the worst rankings on most benchmark datasets. On the other hand, Table 3 demonstrates that the GSA and/or gsaINknn classifiers achieved some of the best recognition results on the testing datasets. Table 2 and especially Table 3 demonstrate the typically poor performance of the SVM classifier.

Table 4 summarizes the overall (average) classification accuracy of every classifier over the 12 benchmark classification datasets considered in this work. More specifically, each cell in Table 4 displays the average of a column of either Tables 2 or 3 regarding the training datasets and the testing datasets, respectively. In addition, each cell in Table 4 displays (within parentheses) the corresponding overall ranking of each classifier. Table 4 indicates that the kNN classifier has demonstrated the best overall accuracy (ranking 1) on the training datasets and a bad accuracy (ranking 7) on the testing datasets. The proposed gsaINknn classifier performed rather poorly on the training datasets (ranking 5); nevertheless, it outperformed all other classifiers on the testing datasets (ranking 1). In other words, the proposed gsaINknn classifier has demonstrated a better overall capacity for generalization than any other classifier on the datasets considered here.

Table 5 sums up the rankings of each classifier for each benchmark classification dataset regarding both training (Table 2) and testing (Table 3). In addition, each Table 5 cell displays (within parentheses) the overall ranking of each classifier. In the aforementioned sense, Table 5 indicates that the best two classifiers in training are the kNN classifier (ranking 1) and the GRNN classifier (ranking 2). Nevertheless, both aforementioned classifiers performed poorly in testing with rankings 7 and 6, respectively. The GSA classifier achieved a moderate overall ranking of 4 in training and an improved overall ranking of 2 in testing. The INkNN classifier achieved the worst overall ranking (8) in training and a moderate overall ranking (4) in testing. Finally, the proposed gsaINknn classifier has demonstrated a moderate overall ranking (5) in training and the best overall ranking (1) in testing.

Next, we examined the impact of different parameter initializations regarding the proposed gsaINknn classifier. Table 6 indicates the impact of different parameter initializations on the classification results as well as on the CPU processing times of the gsaINknn classifier.
classifier regarding three different benchmark datasets. More specifically, different initial values of the parameters $N_a$ (number of agents), IN weight and $G_0$ (gravitational constant), shown in columns two, three and four of Table 6, were employed resulting in the train and test classification accuracies (%) shown in the next two columns of Table 6, respectively. The last column of Table 6 displays the corresponding CPU processing times (in ms). Our experience with different parameter initializations for the gsaINknn classifier suggested that the best training/testing classification accuracies were typically obtained for smaller "IN weight" values (hence for larger number $N_a$ of agents), whereas the initial value $G_0$ of the gravitational constant does not affect significantly the classification accuracies and not at all the CPU processing times as shown in Table 6.

Table 7 displays the CPU processing times (in ms) of all the classifiers on all the benchmark datasets used in this work. We remark that a processing time shown in Table 7 includes both the time for training and the time for testing. It turns out that, even though the knn and GRNN classifiers require no training, fuzzy-ART is the fastest classifier here because fuzzy-ART mostly computes minima. The SOM and SVM classifiers are fairly fast because they carry out specific vector data processing. Likewise,
the INknn classifier is fairly fast. Nevertheless, both the GSA and gsaINknn classifiers require substantially more time because they carry out stochastic search. More specifically, the number of agents employed by either classifier GSA or gsaINknn increases in proportion to the number of instances in a benchmark dataset thus leading to longer training times. Typically, the GSA is faster than the gsaINknn because the GSA does not involve INs. However, the gsaINknn may reduce the number of its INs, hence it may achieve smaller processing times. For example, for the Credit Approval dataset in Table 7, note that the GSA and gsaINknn classifiers have employed 240 and 10 agents resulting in processing times of 1,066,730 [ms] and 447,616 [ms], respectively.

Another point of interest regards the effectiveness of the GSA method. Note that alternative stochastic search and optimization methods, namely genetic algorithms, have been employed with an INknn classifier, and similar classification improvements have been reported. More specifically, genetic algorithms have improved the performance of INknn classifiers by estimating optimally the parameters of both positive valuation functions and dual isomorphic functions (Kaburlasos and Papadakis, 2006; Tsoukalas et al., 2013) in classification experiments regarding the Iris and the Wine benchmark datasets. We point out that any differences between the classification results reported in Kaburlasos and Papadakis (2006), Tsoukalas et al. (2013) and the corresponding results reported here (regarding the Iris and the Wine benchmark datasets) are not statistically significant. Nevertheless, a detailed comparison of alternative optimization methods for the INknn classifier is a topic for future research.

In conclusion, INs have demonstrated here comparatively a significant potential in classification applications, which (potential) is attributed to the fact that an IN can represent all-order data statistics (Kaburlasos et al., 2013a). In addition, the computational experiments in this work have demonstrated that INs without optimization result in a rather poor classification accuracy as shown in both Tables 4 and 5 for the INknn classifier; whereas, an optimization (pursued here by the GSA) of the INknn classifier has significantly improved the classification accuracy.

7. Conclusion

This work has presented an optimized, granular knn classifier (with \(k = 1\)) in the metric lattice of Intervals’ Numbers (INs). Optimization was justified on the grounds of improving the classification results. Since there are no analytic optimization methods currently available in the metric lattice of INs, we resorted to stochastic optimization techniques. In particular, this paper introduced a synergy, namely gsaINknn, of the gravitational search algorithm (gsa) for stochastic optimization with the INknn classifier. An application on 12 benchmark classification datasets from the literature has demonstrated that the gsaINknn classifier achieved better results than either the GSA or the INknn classifier alone. Moreover, our experimental results have demonstrated that the proposed gsaINknn classifier compares favorably with alternative classifiers from the literature.

Potential future work includes the consideration of more than one (N-tuple) IN per class as well as the employment of alternative tunable non-linear functions \(v(x)\) and \(\theta(x)\) rather than the functions \(v(x) = x\) and \(\theta(x) = 1 - x\) employed in this work. Alternative optimization methods to GSA will also be considered comparatively in a future work. The IN inputs to the gsaINknn classifier in this work have been trivial. However, Proposition 1 has paved the way for accommodating non-trivial IN inputs toward dealing with input uncertainty and/or ambiguity. Further future work extensions include the employment of Type-2 INs toward Computing With Words (Mendel, 2007).

References
