Contents lists available at ScienceDirect

Pattern Recognition Letters

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

From neighbors to strengths - the k-strongest strengths (kSS) classification algorithm☆

Juan Aguilera[a], Luis C. González[b,*], Manuel Montes-y-Gómez[a], Roberto López[b], Hugo J. Escalante[a]

[a] Instituto Nacional de Astrofísica, Óptica y Electrónica, Tonantzintla, Puebla 72840, México
[b] Universidad Autónoma de Chihuahua, Circuito Universitario Campus II, Chihuahua, Chih 31100, México

ARTICLE INFO

Article history:
Received 11 March 2019
Revised 21 May 2020
Accepted 22 June 2020
Available online 23 June 2020

MSC:
41A05
41A10
65D05
65D17

Keywords:
Classification problems
k-Nearest neighbor
Gravitational force

ABSTRACT

In this study we introduce the k-Strongest Strengths (kSS) Classification Algorithm, a novel approach for classification problems based on the well-known k-Nearest Neighbor (kNN) classifier. The proposed kSS method is motivated by an analogy to the Law of Universal Gravitation. The novelty of kSS resides in that instead of only using the neighbors' labels to classify an unseen object it uses *gravitation forces* or strengths exerted by training objects. To incorporate this Newtonian concept into kSS, mass to training objects needs to be assigned. Following this idea we propose novel mass functions that exploit object's topology properties within 11 data sets, which comprise binary and multi-class problems that present high and low imbalance ratio. Experiments show that kSS obtains an average f_1 score of 0.87, outperforming other popular Machine Learning methods such as Artificial Neural Networks (0.80), Decision Trees (0.75), other kNN variants (0.79), naïve Bayes (0.72) and Support Vector Machines (0.82). All these differences being statistically significant.

1. Introduction

The k-Nearest Neighbors (kNN) algorithm has become an obligated baseline when dealing with classification and regression tasks, two common problem formulations in the field of Pattern Recognition. In the practice, this is justified by the fact that it is a very intuitive algorithm, and at the same time very effective and capable of obtaining competitive results. Likewise, kNN is theoretically appealing because it has been shown that, for a large enough number of samples, it converges to an error rate at worst twice the error of a Bayes classifier [5]. Nowadays, kNN is currently envisioned for trendy and challenging scenarios such as large-scale learning [20], Big Data analysis [12], and deep learning [13].

A first analysis on the philosophy in which kNN is based suggests that we as humans have hardwired some of the same mechanisms that this algorithm uses, since in many situations we react in the way that we have reacted in similar contexts, thus applying the so-called classification-by-analogy principle [8]. The first ideas to formalize this algorithm were documented by Fix and Hodges [10], where the authors attempted to classify unseen objects by means of the kNN rule, i.e., the class was defined as the class of the majority of the *k* nearest training objects. From those seminal ideas, some variations with the aim of adding more context to the voting strategy have been envisioned, e.g., weighting the class of the kNN by their respective distance to the testing object [9]. About the same time, a clustering algorithm mimicking physical gravitation was proposed [23], and it was a matter of time for the community to realize that the kNN algorithm could be boosted by simulating physical forces such as gravitation [21]. Most works in this path have followed two main ideas to enhance the voting strategy of kNN variants. The first group is composed by those works that propose to assign a mass to objects in the training set [1,15,19,22,24]. While the second group has attempted to modify the way that distances (seen as influences) among objects are calculated [4,16].

In this study, we propose a novel variation of the kNN algorithm that aims to enrich the influence of neighbors by assigning them a mass. With these masses we look to model gravitational forces, or strengths (more precisely), exerted by big bodies, as postulated by Isaac Newton. Then, the class of an unseen object would be the one that corresponds to the largest aggregation of strengths of objects of the same class. We call this approach **The k-Strongest Strengths Classification Algorithm**, or *kSS*. The originality of kSS is supported by the integration of novel mass functions that dra-

☆ Editor: Jayanta Mukhopadhyay
* Corresponding author.
E-mail address: lcgonzalez@uach.mx (L.C. González).

https://doi.org/10.1016/j.patrec.2020.06.020
0167-8655/© 2020 Elsevier B.V. All rights reserved.

matically improve classification performance. Following this idea, we propose four mass functions that exploit knowledge from the topology of training data. Afterwards, for the first time, we show that a Genetic Programming (GP) procedure could be used to optimize the role of data-topology properties into novel expressions used as mass functions.

For the experimental section we robustly evaluate kSS using 11 data sets that present different levels of imbalance ratio, binary and multi-class scenarios and also a varied number of features. We contrast the results of kSS, first, against itself when using specific mass functions; and also against 10 other Machine Learning (ML) classifiers including: four variations of kNN, a Support Vector Machine (SVM), a Decision Tree (DTree), a Naïve Bayes approach (NBayes), a Multi-Layer Perceptron (MLP), and two other state-of-the-art algorithms. All these results statistically validated by the non-parametric Friedman test [6] and later by the non-parametric Bayesian Signed-Rank (BSR) test [3]. Finally, we elaborate on what particular topology properties are more useful to construct highly discriminative mass functions.

The organization of this study is as follows. In Section 2 we visit relevant work that implement gravitation principles within the kNN framework. Section 3 introduces the kSS algorithm and presents novel mass function, created, first, by a hand-craft approach and latter applying a Genetic Programming search procedure. Section 4 presents experiments and results of kSS, while Section 5 discusses mass functions characteristics and properties. Section 6 concludes this study.

2. Literature review

This review is focused on those works that apply gravitational principles to enhance classification performance in nearest-neighbor based algorithms. For example, Peng et al. [15] proposed the Data Gravitation Classification (DGC) algorithm that computes gravitational forces among objects to classify. Results showed that DGC is more efficient than regular kNN since it groups original objects into clusters, where the mass of each cluster is based on the number of objects that it agglutinates. Later, an improvement over DGC, called DGC+, was presented by Cano et al. [4]. DGC+ employs a matrix of weights to assign relative importance to object's attributes depending on the class that it belonged. Following this, the distance metric would be overly influenced by specific attributes depending on each class. DGC+ was evaluated over 79 data sets, both standard and imbalanced, achieving an average accuracy of 0.83. Inspired by these two previous studies, Peng et al. [16] proposed another algorithm called imbalanced DGC model (IDGC). This algorithm alters the way the gravitation force is computed, using an amplified gravitation coefficient (AGC). This value enhances the gravitational pull of the weak class and decreases the gravitational field of the stronger class to compensate for imbalance. IDGC was favorably compared to other algorithms, including kNN and SVM, however, a shortcoming of IDGC resides in its high computational cost when applied to large data sets.

Zhu et al. [24] proposed the Gravitational Fixed Radius Nearest Neighbor (GFRNN) algorithm which was mainly designed to address the issue of class imbalance. GFRNN defines a fixed radius from where it selects the nearest neighbors for a given test object. The chosen objects, then, vote based on the gravitation force law, where all objects of the same class have the same mass, which was calculated in accordance to the imbalance ratio. Later, Shabani-kordshooli et al. [19] proposed to enhance the GFRNN algorithm by using the Gravitational Search Algorithm (GSA), which is employed to search for better masses to objects in the training set. This strategy achieved an average G-Mean score of 0.87 (over 15 data sets), an slight improvement over the original GFRNN. A shortcoming of these two GFRNN-based proposals, is that their algorithms were developed to work on data sets that contain only two classes of objects (focusing only on binary classification). Rezaei and Nezamabadi-pour [17] presents a prototype generation method that uses GSA to determine the best set of prototypes. Wen et al. [22] presents a Cognitive Gravitational Model (CGM) that was based on both the gravitation and the cognitive laws, where the “self-information” of each object, similar to a mass, is computed. This self-information is defined by a density function. In particular, the instances closer to others (regardless their class) have greater density, and therefore, lower self-information. Accordingly, it gives more weight (mass) to instances that are far from the rest. Just recently, Aguilera et al. [1] evaluated two contrasting ideas to assign mass to objects in the training data set. The first idea is to give more mass to those objects surrounded by objects of its same class (CC), whereas the second approach was intended to assign more mass to objects surrounded by objects of different class (CD). The authors tested both ideas within a weighting scheme called Weighted Attraction Force (WAF), outperforming results obtained by the traditional kNN and weighted kNN.

Based on this review, we can observe that some works either present a rigid scheme to find neighbors (fixed radius) or only consider very specific characteristics of the data sets to build mass functions (imbalance ratio). Some others are less general, since they are only evaluated on binary classification problems. In this study we propose a more robust and general algorithm which integrates a search method to look to assign masses that help discriminate the most among classes, and also to be applied not only to binary cases but to multi-class classification too.

3. The k-strongest strengths classification algorithm (kSS)

The design behind kSS relies on the hypothesis that better classification performance could be achieved if objects in the training set are assigned a mass, related to their importance thus mimicking physical attraction forces exerted by big bodies. The pseudo-code of kSS is presented in Algorithm 1. Compared to existing

Algorithm 1 The kSS classification algorithm.

```
Input: Test object to classify $\tilde{x}_q$, training set $X = \{x_1, x_2, \ldots, x_n\}$
Output: Class of object $\tilde{x}_q$ ($f(\tilde{x}_q)$)
1:  //Compute mass for every training object $x_i \in X$.
2:  for i = 1 to n do
3:      Assign a mass to training object $x_i$
4:      //Any of the proposed mass functions could be used
5:  end for
6:  //Compute attraction force between $\tilde{x}_q$ and every $x_i \in X$
7:  for i = 1 to n do
8:      $strength(\tilde{x}_q, x_i) = \frac{m(x_i)}{dist(\tilde{x}_q, x_i)^2}$
9:  end for
10: Find the subset $ss_{\tilde{x}_q} \subseteq X = \{x_1, x_2, \ldots, x_k\}$ of k training objects
    that exert the strongest forces on $\tilde{x}_q$.
11: Determine $f(\tilde{x}_q)$: $f(\tilde{x}_q) \leftarrow \arg\max_{c \in Class} \sum_{i=1}^{k} \delta(c, x_i)$
12: Return $f(\tilde{x}_q)$
```

methodologies, the proposed kSS is not dependent on tailored features only available in certain data sets; also, the classification scheme is flexible and does not assumes a fixed radius for obtaining neighbors.

kSS algorithm remarks

Note that in the kSS algorithm, instead of considering just a distance to establish the neighborhood for a new test object, an attraction force must be computed (steps 7–9). We do this by means of the Law of Universal Gravitation proposed by Isaac Newton in 1687. This law tells that the attraction force between two objects

Table 1
Topology-based mass functions (TopMF) proposed in this study.

Mass function	Equation	Goal
Circled by its own class (CC)	$m(x) = \log_2(SN_k(x, class_x) + 2)$	Highlights objects mostly surrounded by objects of its same class.
Weighted Per Class (WPC)	$m(x) = \log_2(\frac{M}{n_x} + 1)$	Compensate class imbalance.
Cohesion (COH)	$m(x) = \frac{1}{\log_2(dist(x,S_x))}$	Rewards objects that are in the center of its class.
Separation (SEP)	$m(x) = \frac{1}{\log_2(dist(x,D_x))}$	Rewards objects near to frontiers.

is proportional to the product of their masses and inversely proportional to the square distance between them, as expressed by Eq. (1).

$$F = G\frac{m_1 m_2}{d^2} \tag{1}$$

where F is the attraction force, G is the gravitational constant, m_1 and m_2 are the objects' masses and d is the distance between the centers of their respective masses. Based on Eq. (1), we can derive Eq. (2) for kSS.

$$strength(\tilde{x}_q, x_i) = G\frac{m(\tilde{x}_q)m(x_i)}{dist(\tilde{x}_q, x_i)^2} \simeq \frac{m(x_i)}{dist(\tilde{x}_q, x_i)^2} \tag{2}$$

where $strength(\tilde{x}_q, x_i)$ represents the attraction force between the test object $\tilde{x}_q$ and training object x_i, $m(\tilde{x}_q)$ and $m(x_i)$ represent their respective masses, and $dist(\tilde{x}_q, x_i)$ is a function that returns the distance between objects $\tilde{x}_q$ and x_i in the feature space, e.g., euclidian. Note that $m(\tilde{x}_q)$ and G can be omitted given that they are only used as scaling factors. Simplifying, we obtain the expression on the right side, which is the one used in step 7 of the kSS algorithm. Once the attraction force between each training object and test object is computed, just the k strongest forces are considered to assign a label (step 10), which will be the class with the most votes (step 11). In this step, $\delta(c, x_i)$ is a function that returns 1 if x_i belongs to class c and returns 0 otherwise.

3.1. *Proposed mass functions*

To evaluate different ideas to construct mass functions we followed two approaches. The first approach is based on exploiting topology properties of all objects in the training set, e.g., objects' centrality, closeness to other objects of the same class, etc. We called this approach Topology-based Mass Functions, or for simplicity, *TopMF*. The second approach attempts to systematize the creation of mass functions based on effectively searching of better ways to harmoniously integrate the so-called TopMF. This process is carried out by the implementation of a Genetic Programming (GP) strategy.

3.1.1. *Topology-based mass functions (TopMF)*

The guiding hypothesis to design these mass functions is that spatial distribution or topology of the classes in the training set can be exploited in favor of certain objects. Following this idea, we propose four mass functions which are listed in Table 1. For these formulations x corresponds to the training object that aims to be assigned a mass, and the function $dist(x, Set)$ returns the sum of all distances between training object x and objects that belong to subset *Set*.

- Circled by its own class (**CC**). For CC, $class_x$ represents the class of object x and the function $SN_k()$ returns how many of the k closest objects to x are of the same class of x. The $\log_2()$ function serves as a smoothing factor, and the constant 2 is added to avoid computation errors.
- Weighted per class (**WPC**). For this formulation, M represents the number of objects of the majority class and n_x the number of objects of class c_x for training object x.

Table 2
Input variables for the GP process (x_i represents a training object).

Variable	Component	Description
pc_1	$\frac{n_{maj}}{n_{\min}}$	Imbalance ratio.
pc_2	n_{x_i}	# objects in the class of x_i.
pc_3	n_{maj}	# objects in the majority class.
pc_4	N	Total number of training objects.
pc_5	$dist(< x_i, f(x_i) >, C_{f(x_i)})$	Distance from x_i to its centroid.
pc_6	$dist(< x_i, f(x_i) > , C_{f(y)})$	Distance from x_i to the nearest centroid of different class.
pc_7	$\Sigma dist((< x_i, f(x_i) > , S))$	Sum of the distances from x_i to the rest objects of its same class.
pc_8	$\Sigma dist((< x_i, f(x_i) > , D))$	Sum of the distances from x_i to the objects of different class.
pc_9	$ISN(x_i)$	# immediate objects to x_i of its same class.
pc_{10}	$IDN(x_i)$	# immediate objects to x_i of different class.

- Cohesion (**COH**). Here S_x corresponds to the training set of objects that have the same class as object x.
- Separation (**SEP**). For this formulation, D_x represents the set of training objects with a class different to the class of object x.

3.1.2. *Mass functions generated by genetic programming*

Genetic Programming (GP) is a popular and robust search strategy that has been successfully employed to synthesize mathematical expressions to describe a phenomenon of interest. Inspired by nature, GP evolves individuals (solutions) based on a given fitness function and selection pressure [11]. Therefore, we propose the use of GP for automatically obtaining mass functions that can enhance the performance of those proposed in Section 3.1.1. The hypothesis is that GP can discover better ways of combining primitives used in the definition of masses for generating better mass functions driven by the performance of kSS. For this, we propose that the input for GP be the components used to construct the TopMF. Table 2 presents the list of components, and their description, that are going to be used as GP's input variables (pc_i) in the evolutionary process, see below.

For reference, Algorithm 2 presents the canonical Genetic Programming formulation considered in this study, specifically, the GPTips implementation was used [18]. GP relies on a population of

Algorithm 2 Genetic programming.

```
1:  for $i$ = 1 to Generation $M$ do
2:      if 1st generation then
3:          Generate initial population
4:      else
5:          Apply crossover and mutation to generate a new population
6:      end if
7:      Calculate fitness of each individual within the population
8:      Select n members from population (tournament parameters)
        to participate in crossover and mutation
9:  end for
10: Return best individual in last population
```

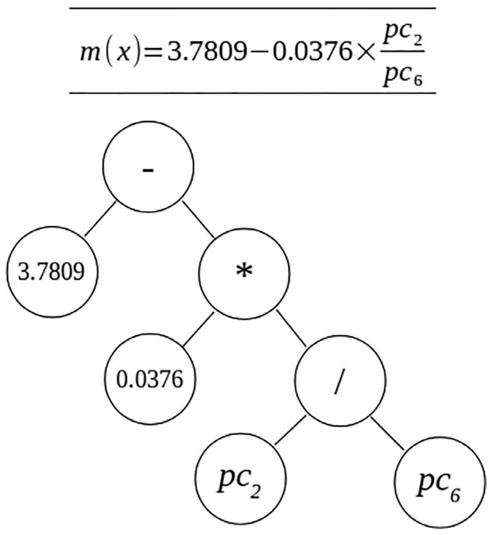

Fig. 1. Tree representation of an individual within GP.

solutions encoded in syntactic trees, solutions are evolved (modified) in an iterative process of M steps. To generate an individual, input variables are randomly (and coherently) combined via a set of mathematical operators, e.g., +, /, sin, and constants. Input variables are used as leaves and operators as intermediate nodes. Consider Fig. 1, where an example of an individual (also known as *chromosome*) is represented by a syntactic tree. Observe that an individual provides a *recipe* to compute a value, which in this context means a way to calculate masses for training objects.

Once the population is generated, a fitness value (assessing the *goodness* of solutions) is computed for each individual. To guide the evolution of the population towards good mass functions, we incorporated the kSS classification performances as a feedback to GP via the fitness function. That is, each mass function induced from a syntactic tree was used within the kSS (step 7 of Algorithm 2) to evaluate its classification performance. The f_1 classification score obtained by each mass function was then used as the individual's fitness value. The probability of selecting an individual to produce offspring is directly proportional to its fitness. Based on these probabilities, n individuals are chosen to participate in crossover. To generate a new individual a tournament selection protocol is employed to pick two parents. Then, randomly chosen sub-trees from each of these two parents are merged, generating a new individual. For the case of mutation, a chosen individual is modified by randomly eliminating one of its sub-trees or by replacing one of its mathematical operator. All this process is iterated for M generations, yielding at the end an evolved solution.

4. Experiments and results

To conduct the experimental evaluation of kSS we selected 11 data sets from the UCI repository [7]. This data only contain numerical values as attributes and there are no missing values. Table 3 presents a detailed description of the data sets and some of their characteristics, which are diverse in a number of ways, e.g., in the number of instances, number of classes, and the class imbalance ratio (IR). For the classification process a 10-fold cross-validation was applied and the classification performance by different approaches is measured in terms of f_1 score.

GP default parameters were used for our experiments [11]: population size is 100 individuals evolving through 100 generations, and the operators and constants sets are composed by: $+, -, /, *, log_2, |x|, \sqrt{x}, x^2, x^y$, and random numbers $\in$ [1,10], respectively.

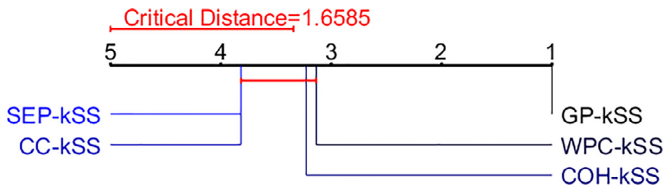

Fig. 2. CD diagram for the comparison of kSS using the four TopMF and GP-based mass functions.

The first experiment approaches the question of how handcrafted TopMF mass functions compare to each other. For this we evaluate mass functions CC, WPC, COH and SEP within the kSS algorithm. Moreover, to add some context to this comparison we run the GP search and include into this first experiment the mass function that was obtained after the search process. For all comparisons, and based on preliminary experimentation, k=7 for any version of the k-nearest neighbors algorithm (included kSS).

Table 4 presents f_1 scores obtained by each kSS (using a particular MF) over all data sets. From this result we observe that the GP-based mass functions clearly boosts the classification performance of kSS, even in some data sets with a dramatic gain of more than 20% (and more than 10% on average) with respect to the second best MF, which in this case is the Weight Per Class (WPC).

To validate these differences we applied the non-parametric Friedman test [6] with a significance level of $\alpha = 0.05$.

To visually help in the interpretation of this analysis we use a Critical Difference (CD) diagram [6]. In this diagram all methods are ordered based on their average rank, where the best method is located rightmost. The CD diagram joins with a thick horizontal line those methods for which there is no evidence of statistical difference, i.e., whose difference in average rank is less than the critical distance. The CD diagram of the first experiemnt is shown in Fig. 2. Now the interpretation is clear, the results show that kSS (from here on called GP-kSS) using the GP-based mass functions is statistically better than kSS using any of the other four TopMF mass functions.

Given that GP-kSS proved to be the strongest classification algorithm in the previous experiment, the remaining question is: how it would compare against other more established ML classifiers? In this regard, we include in this comparison four kNN-based algorithms: kNN, Distance Weighted kNN (DWkNN) and the Weighted Attraction Force algorithm (WAF) with two mass functions (CC and CD) as proposed by Aguilera et al. [1]. We also incorporate into this evaluation a Decision Tree (max depth = 7), a Naïve Bayes classifier (Gaussian), a Support Vector Machine (kernel = RBF, C = 8.5) and a Multi-Layer Perceptron (hidden layer sizes = {12,12,12}, activation function = relu, max iterations = 500), all implementations provided by the Python-based suite Scikit-Learn [14].[1] To make this experiment more comprehensive we also consider two state-of-the-art approaches specifically tailored to use gravitation principles as well: GFRNN [24] and Im-GFRNN [19].

Table 5 presents the results of this experiment. After analyzing these results, we observe that GP-kSS keeps showing an outstanding performance, well beyond the kNN based variants, and even yielding better results than popular SVM and MLP. In a direct comparison against state-of-the-art approaches, it surpasses them by more than 10% (GFRNN) and 25% (Im-GFRNN).

We follow the same statistical analysis as in the previous experiment to validate differences in performances for all approaches. For this analysis the CD diagram is shown in Fig. 3. According to this test, GP-kSS is different than most of the ML approaches, ex-

[1] To select the best hyper-parameters for all methods a Grid search procedure was implemented.

Table 3
Data sets characteristics.

Data sets	Instances	Features	Classes	Classes distribution	IR
Ecoli	336	7	8	143/77/52/35/20/5/2/2	71.50
Glass	214	9	6	76/70/29/17/13/9	8.44
Haberman	306	3	2	225/81	2.78
Ionosphere	351	34	2	225/126	1.79
Iris	150	4	3	50/50/50	1.00
Pima	768	8	2	500/268	1.87
Sonar	208	60	2	111/97	1.14
Thyroid	215	5	3	150/35/30	5.00
Vehicle	846	18	4	218/217/212/199	1.10
WDBC	569	30	2	357/212	1.68
Wine	178	13	3	71/59/48	1.48

Table 4
F_1 scores for the comparison among versions of kSS when considering TopMF and GP-based mass functions.

Data sets	CC-kSS	WPC-kSS	COH-kSS	SEP-kSS	GP-kSS
Ecoli	0.7483	0.7276	0.7064	0.7527	**0.9082**
Glass	0.4881	0.5322	0.5671	0.5301	**0.7463**
Haberman	0.5638	0.5760	0.5322	0.5220	**0.6589**
Ionosphere	0.7635	0.9258	0.8155	0.6847	**0.9726**
Iris	0.9677	0.9603	0.9467	0.9467	**0.9943**
Pima	0.7174	0.6672	0.7095	0.6698	**0.7879**
Sonar	0.7333	0.8193	0.8277	0.7481	**0.9115**
Thyroid	0.8310	0.9411	0.9065	0.8235	**0.9698**
Vehicle	0.6550	0.7168	0.7100	0.7087	**0.7598**
WDBC	0.9617	0.9321	0.9652	0.9615	**0.9750**
Wine	0.9568	0.9118	0.9316	0.9730	**0.9854**
MEAN	0.7624	0.7918	0.7835	0.7564	**0.8791**

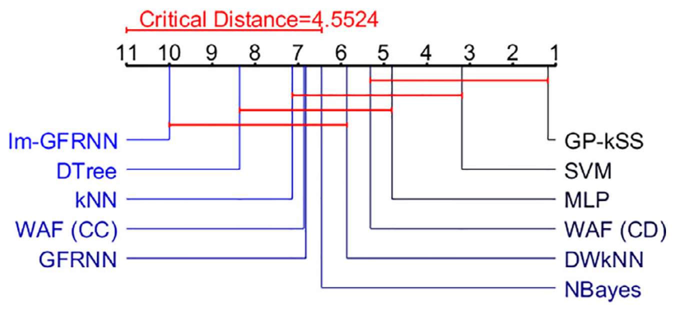

Fig. 3. CD diagram for the comparison of GP-kSS against 10 other ML classifiers.

cept for SVM, MLP and WAF(CD), so our next analysis aims to compare in a finer detail kSS against the second best classifier, SVM.

To compare side by side kSS and SVM, the two best classifiers in this study, we will make use of the non-parametric Bayesian Signed-Rank (BSR) test [3]. BSR directly answers the question of given two treatments which one is better, even giving a probability to support that claim. That is, BSR shows the probability of occurrence of three different scenarios when applying two algorithms A and B over a given data set. These scenarios are the following: (scenario 1) A outperforms B; (scenario 2, called *rope* or region of practical equivalence) A and B have similar performance; and (scenario 3) B outperforms A. Table 6 shows the probabilities obtained by BSR for the comparison of SVM (treatment A) and GP-kSS (treatment B). This bayesian analysis shows that there is a probability of 99.34% for GP-kSS to be superior to SVM.

Table 6
BSR output probabilities.

Compared algorithms	Scenarios		
	(1) $A > B$	(2) $A = B$	(3) $A < B$
SVM vs GP-kSS	0.0063	0.0003	**0.9934**

To help visualize this analysis, we map 150,000 Monte Carlo samples in barycentric coordinates as proposed by Benavoli et al. [2], where each vertex of the triangle is associated to each BSR scenario, left (scenario 1), right (scenario 3), and top (scenario 2). Fig. 4 shows what was previously computed as probability, GP-kSS resulting in the best classifier.

5. Discussion

Having observed the competitive performance of kSS when using GP evolved mass functions, we would like to further analyze some characteristics of the evolved formulae. Table 7 shows some of the best mass functions found per data set and the f_1 score that they obtained for a given particular fold. Most of these mass functions are simple mathematical expressions that only include one or two input variables, but in such a fashion that they are capable of assigning highly discriminative masses to training objects. Further,

Table 5
F_1 scores for the comparison of GP-kSS against 10 other ML classifiers.

Data sets	kNN	DWkNN	WAF (CC)	WAF (CD)	GFRNN	Im-GFRNN	DTree	NBayes	SVM	MLP	GP-kSS
Ecoli	0.7482	0.7465	0.7517	0.7251	0.6108	0.2354	0.6573	0.5042	0.7562	0.7354	**0.9082**
Glass	0.5358	0.5724	0.5773	0.5940	0.5653	0.2856	0.5663	0.4080	0.5816	0.5602	**0.7463**
Haberman	0.5235	0.5218	0.5098	0.5296	0.5971	0.4566	0.5863	0.5907	0.5434	0.5445	**0.6589**
Ionosphere	0.7773	0.7773	0.7851	0.8306	0.5878	0.4043	0.8588	0.8656	0.9418	0.9091	**0.9726**
Iris	0.9467	0.9678	0.9539	0.9539	0.9526	0.9674	0.9455	0.9381	0.9554	0.9522	**0.9943**
Pima	0.7039	0.7065	0.6941	0.6949	0.7207	0.3937	0.6818	0.6443	0.7059	0.7200	**0.7879**
Sonar	0.8117	0.8165	0.8033	0.8392	0.8045	0.5845	0.7059	0.7104	0.8769	0.8096	**0.9115**
Thyroid	0.8772	0.9154	0.9094	0.9162	0.9352	0.3241	0.8742	0.8882	0.9246	0.9224	**0.9698**
Vehicle	0.7109	0.7073	0.6881	0.7197	0.6054	0.6315	0.6749	0.7138	**0.8256**	0.8097	0.7598
WDBC	0.9612	0.9612	0.9612	0.9615	0.9400	0.5195	0.9070	0.9271	0.9736	0.9652	**0.9750**
Wine	0.9635	0.9635	0.9635	0.9686	0.9303	0.8978	0.8889	0.9722	0.9749	0.9748	**0.9854**
MEAN	0.7782	0.7869	0.7816	0.7939	0.7500	0.5182	0.7588	0.7258	0.8236	0.8094	**0.8791**

Table 7
Examples of mass functions generated by GP for each data set in a single fold.

Data set	Mass function	F_1 score
Ecoli	$3.2749 - 107.0172 \times \frac{(pc_5+pc_9)^{0.5}}{pc_4}$	0.9634
Glass	$\frac{6.0104E15 \times pc_9^{0.5} \times (pc_3 - pc_7)}{2.8823E17 \times pc_6 + 5.3608E17} + 3.7029$	0.8786
Haberman	$1.3845 - 8.5856E{-7} \times \frac{pc_7^2}{pc_6^2}$	0.7801
Ionosphere	$0.0115 \times pc_{10} + 0.3545$	1.0000
Iris	$0.5095 \times pc_5 - 0.0089 \times pc_8 + 3.8569$	1.0000
Pima	$0.0027 \times pc_1 + 0.0471 \times pc_6 \times pc_9 - 4.6357E{-8} \times pc_7^2 \times pc_9 + 6.7395E{-8} \times pc_7^2 + 0.0014$	0.8396
Sonar	$1.4015 - 0.0269 \times pc_8^{1/2}$	1.0000
Thyroid	$2.8450 - 0.0133 \times pc_2$	1.0000
Vehicle	$2.5963 - 0.1954 \times pc_{10}$	0.8073
WDBC	$0.8835 - 0.0638 \times pc_5$	1.0000
Wine	$0.9508 \times pc_1 + 0.7208$	1.0000

Table 8
Generalization results in terms of f_1 score of mass functions from Table 7 in all data sets.

		Original mass functions derived from these datasets.										
		Ecoli	Glass	Haberman	Iono	Iris	Pima	Sonar	Thyroid	Vehicle	WDBC	Wine
Results in these data sets.	**Ecoli**	0.7233	0.5258	0.7452	0.7016	0.0528	0.5940	0.7416	0.7292	0.7446	0.7395	0.7482
	Glass	0.5986	0.5478	0.5698	0.5841	0.0984	0.5733	0.5194	0.6016	0.5247	0.5338	0.5358
	Haberman	0.3840	0.3215	0.5828	0.5322	0.4566	0.5157	0.4958	0.3552	0.5586	0.5357	0.5235
	Iono	0.8944	0.4043	0.9158	0.9244	0.4043	0.8303	0.4607	0.6968	0.7662	0.7500	0.7773
	Iris	0.9439	0.9603	0.9467	0.9467	0.9205	0.8600	0.9467	0.9467	0.9467	0.9467	0.9467
	Pima	0.2722	0.2574	0.6753	0.6769	0.3937	0.6800	0.3933	0.2574	0.7035	0.7036	0.7039
	Sonar	0.7930	0.6799	0.8378	0.8058	0.3435	0.7232	0.6940	0.8046	0.7797	0.7836	0.8117
	Thyroid	0.9230	0.5566	0.9309	0.8772	0.2728	0.6563	0.8186	0.9411	0.8772	0.8610	0.8772
	Vehicle	0.6582	0.1021	0.7115	0.7216	0.2043	0.6810	0.1659	0.6087	0.7061	0.6999	0.7109
	WDBC	0.9523	0.9316	0.9451	0.9613	0.3833	0.9519	0.4147	0.2697	0.9596	0.9573	0.9612
	Wine	0.9422	0.8624	0.9255	0.9635	0.1868	0.9781	0.9667	0.9316	0.9568	0.9635	0.9635
	Avg	0.735	0.559	0.7987	0.7904	0.3379	0.7312	0.6015	0.6493	0.7748	0.7704	0.7781

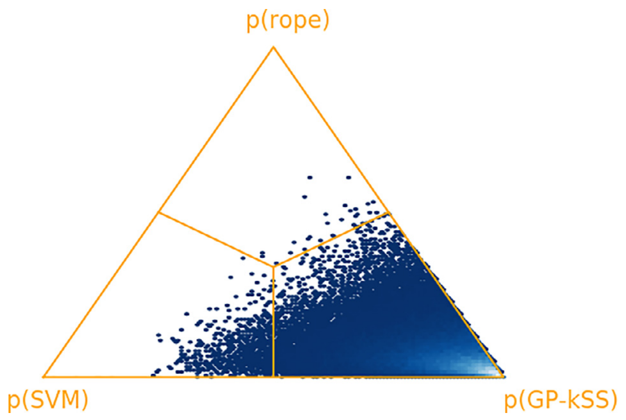

Fig. 4. Visualization of the output probabilities from the BSR test.

Table 9
Variables involved in the complexity analysis.

Variable	Description
N	Training objects
N'	Test objects
M	Attributes (features of objects)
K	Number of k closest neighbors
L	Number of mathematical operations in the mass functions
G	Generations for GP
P	Number of individuals for GP

Table 10
Execution time of kSS in Big O notation.

Algorithm	Complexity	
	Training	Testing
kNN	———	O(MxN)
kSS	O(PxMxNxG)	O(MxN)

Fig. 5 shows a global analysis of the frequency of use of all these variables. Based on this, we observe that there are some primitive components (see Table 2 for the complete list) that are more prone to be considered in the process of building a mass functions, e.g, pc_6 (distance from an object to the nearest different centroid), then suggesting its importance.

Another analysis is the one related to the capacity of mass functions to generalize, that is, how a mass function evolved for a particular data set performs in another data set. To accomplish this task we took all mass functions generated by GP for each data set and evaluate them in all data sets, the results in terms of f_1 score are shown in Table 8. There are some cases where an specific mass function struggles to offer competitive results, for instance the Sonar mass function in the Vehicle dataset ($f_1 = 0.1659$). However, we observe that most of the datasets-specific mass functions obtain acceptable performance, even competing with those scores obtained by traditional ML methods, e.g., NBayes obtained an average $f_1 = 0.7270$ over all data sets whereas the Haberman (GP evolved) mass function registers 0.7987. This finding suggests that not only GP is able to find very competitive mass functions per data set, but also that these mass functions could attain some degree of generalization.

5.1. kSS complexity analysis

In the following we measure the complexity of kSS in Big O notation. Table 9 presents the list of variables considered for the analysis while Table 10 presents the results.

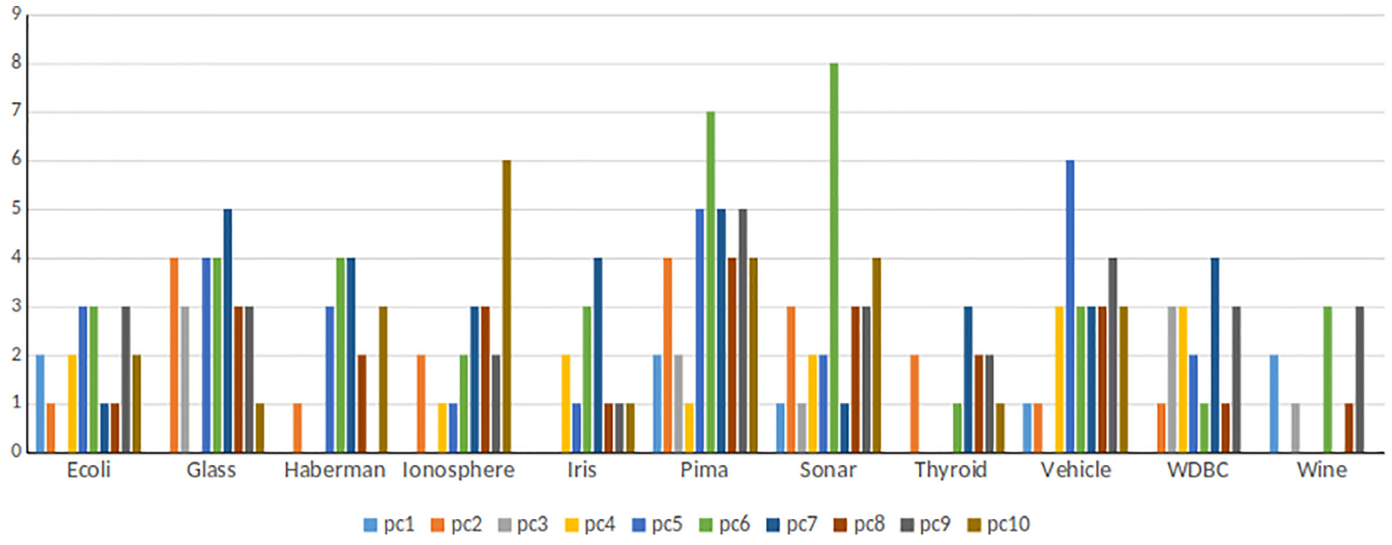

Fig. 5. Variable usage to construct the best mass functions as found by GP.

For testing purposes both algorithms, kNN and kSS have similar complexity. For the training phase, absent in kNN, kSS (through GP) has to optimize mass functions with good capacity for classifying objects, but still investing polynomial time (one should note that this process has to be done a single time, and it is performed offline).

For the case of the topMF the analysis showed that CC is O(MxN), WPC is O(N), COH is O(MxN) and SEP is O(MxN).

6. Conclusion

In this study we propose a novel classification algorithm called *The k-Strongest Strengths Classification Algorithm* or kSS. kSS, based on the kNN algorithm, assigns a label to a unseen object depending on the k strongest forces or strengths exerted by objects in the training set. To boost the classification performance of kSS we introduce in this study novel mass functions originated in two ways: following a Topology-based approach and the one where we employ GP. In a first experiment, whose goal was to determine the best mass functions to use in 11 data sets, results clearly favored those mass functions generated by GP. Later, we contrasted GP-kSS against other ML classifiers (which included traditional kNN versions, a SVM and state-of-the-art approaches), concluding that GP-kSS offered the best classification results, even being statistically significant with respect to most of the classifiers.

In a further experiment we investigated about the nature of GP-based mass functions, which proved to be simple, highly competitive and generalizable among data sets. Thus positioning kSS as a strong baseline competitor for classification problems.

Declaration of Competing Interest

There is no conflict of interest of any type.

Acknowledgments

This research was partially supported by the CONACYT grant FC 2016-01-2410. First author was supported by a CONACyT graduate scholarship with number 620113.

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.patrec.2020.06.020.

References

[1] J. Aguilera, L.C. González, M. Montes-y Gómez, P. Rosso, A new weighted k-nearest neighbor algorithm based on Newton's gravitational force, in: Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications, Springer International Publishing, 2018, pp. 305–313.
[2] A. Benavoli, G. Corani, J. Demsar, M. Zaffalon, Time for a change: a tutorial for comparing multiple classifiers through Bayesian analysis 18 (2016) 1–36.
[3] A. Benavoli, F. Mangili, G. Corani, M. Zaffalon, F. Ruggeri, A Bayesian Wilcoxon signed-rank test based on the Dirichlet process, Proc. 31 Int. Conf. Mach. Learn. 32 (2014) 9.
[4] A. Cano, A. Zafra, S. Ventura, Weighted data gravitation classification for standard and imbalanced data, IEEE Trans. Cybern. 43 (6) (2013) 1672–1687, doi:10.1109/TSMCB.2012.2227470.
[5] T. Cover, P. Hart, Nearest neighbor pattern classification, IEEE Trans. Inf. Theor. 13 (1) (1967) 21–27, doi:10.1109/TIT.1967.1053964.
[6] J. Demŝar, Statistical comparisons of classifiers over multiple data sets, J. Mach. Learn. Res. 7 (2006) 1–30.
[7] D. Dheeru, G. Casey, Machine Learning Repository, School of Information and Computer Sciences, University of California, Irvine, 2017 http://archive.ics.uci.edu/ml.
[8] P. Domingos, The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World, Basic Books, 2015.
[9] S.A. Dudani, The distance-weighted k-nearest-neighbor rule, IEEE Trans. Syst. Man Cybern. SMC-6 (4) (1976) 325–327.
[10] E. Fix, J. Hodges, JL, Discriminatory analysis nonparametric discrimination: consistency properties, USAF School of Aviation Medicine, Randolph Field, TX, Project No. 21-49-004, Rep. No. 4, Contract No. AF41(128)-31(1951).
[11] W.B. Langdon, R. Poli, Foundations of Genetic Programming, Springer-Verlag New York, Inc., New York, NY, USA, 2002.
[12] J. Maillo, S. Ram-rez, I. Triguero, F. Herrera, KNN-is: An iterative spark-based design of the k-nearest neighbors classifier for big data, Knowledge-Based Systems 117 (2017) 3–15, doi:10.1016/j.knosys.2016.06.012. Cited By 64
[13] N. Papernot, P.D. McDaniel, Deep k-nearest neighbors: towards confident, interpretable and robust deep learning, CoRR abs/1803.04765(2018).
[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: machine learning in python, J. Mach. Learn. Res. 12 (2011) 2825–2830.
[15] L. Peng, B. Yang, Y. Chen, A. Abraham, Data gravitation based classification, Inf. Sci. 179 (6) (2009) 809–819, doi:10.1016/j.ins.2008.11.007.
[16] L. Peng, H. Zhang, B. Yang, Y. Chen, A new approach for imbalanced data classification based on data gravitation, Inf. Sci. 288 (C) (2014) 347–373, doi:10.1016/j.ins.2014.04.046.
[17] M. Rezaei, H. Nezamabadi-pour, A prototype optimization method for nearest neighbor classification by gravitational search algorithm, in: 2014 Iranian Conference on Intelligent Systems (ICIS), 2014, pp. 1–4, doi:10.1109/IranianCIS.2014.6802522.
[18] D. Searson, GPTIPS Genetic Programming and Symbolic Regression for MATLAB User Guide, 2009.
[19] M. Shabani-kordshooli, B. Nikpour, H. Nezamabadi-pour, An improvement to gravitational fixed radius nearest neighbor for imbalanced problem, in: 2017 Artificial Intelligence and Signal Processing Conference (AISP), 2017, pp. 262–267, doi:10.1109/AISP.2017.8324109.
[20] P. Skryjomski, B. Krawczyk, A. Cano, Speeding up k-nearest neighbors classifier for large-scale multi-label learning on GPUs, Neurocomputing 354 (2019) 10–19, doi:10.1016/j.neucom.2018.06.095. Recent Advancements in Hybrid Artificial Intelligence Systems

[21] C. Wang, Y.Q. Chen, Improving nearest neighbor classification with simulated gravitational collapse, in: L. Wang, K. Chen, Y.S. Ong (Eds.), Advances in Natural Computation, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, pp. 845–854.
[22] G. Wen, J. Wei, J. Wang, T. Zhou, L. Chen, Cognitive gravitation model for classification on small noisy data, Neurocomputing 118 (2013) 245–252, doi:10.1016/j.neucom.2013.02.033.
[23] W. Wright, Gravitational clustering, Pattern Recognit. 9 (3) (1977) 151–166, doi:10.1016/0031-3203(77)90013-9.
[24] Y. Zhu, Z. Wang, D. Gao, Gravitational fixed radius nearest neighbor for imbalanced problem, Knowl. Based Syst. 90 (2015) 224–238, doi:10.1016/j.knosys.2015.09.015.