Expert Systems With Applications 43 (2016) 9–14

Contents lists available at ScienceDirect

Expert Systems With Applications

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

An evolutionary voting for *k*-nearest neighbours

Daniel Mateos-García[*], Jorge García-Gutiérrez, José C. Riquelme-Santos

Department of Computer Science, Avda. Reina Mercedes S/N, Seville 41012 , Spain

ARTICLE INFO

Keywords:
Evolutionary computation
Nearest-neigbour
Weighted voting

ABSTRACT

This work presents an evolutionary approach to modify the voting system of the *k*-nearest neighbours (kNN) rule we called EvoNN. Our approach results in a real-valued vector which provides the optimal relative contribution of the *k*-nearest neighbours. We compare two possible versions of our algorithm. One of them (EvoNN1) introduces a constraint on the resulted real-valued vector where the greater value is assigned to the nearest neighbour. The second version (EvoNN2) does not include any particular constraint on the order of the weights. We compare both versions with classical kNN and 4 other weighted variants of the kNN on 48 datasets of the UCI repository. Results show that EvoNN1 outperforms EvoNN2 and statistically obtains better results than the rest of the compared methods.

1. Introduction

Weighting in machine learning is a common technique for emphasizing some characteristics of data to improve the resulting models. For example, weighting has been used to outline the importance of some particular instances (Blachnik & Duch, 2011) or features (Zhi, Fan, & Zhao, 2014), or rank a set of techniques in the context of ensembles (Berikov, 2014). In a broad sense, Artificial Neural Networks (ANNs) and Support Vector Machines (SVMs) can be also seen as examples of using weights in learning models but the *k*-nearest neighbours (kNN) has been the most common technique to benefit from weights (Mateos-García, García-Gutiérrez, & Riquelme-Santos, 2012).

kNN and its variants have been widely used in the literature to solve real problems. Rodger (2014) used a hybrid model to predict the demand of natural gas. The system was implemented integrating regression, fuzzy logic, nearest neighbour and neural networks, and considering several variables such as the price, operating expenses, cost to drill new wells, etc. If we focus on biological data, Park and Kim (2015) selected significant genes from microarrays by using a nearest-neighbour-based ensemble of classifiers. On the other hand, Park, Park, Jung, and Lee (2015) tackled the problem of designing recommender systems. For this purpose the authors presented Reversed CF (RCF), a fast item-based collaborative filtering algorithm which utilizes a *k*-nearest neighbour graph.

The main goal of a weighting system lies in the optimization (commonly by metaheuristics) of a set of weights in the training step to obtain the highest accuracy but trying not to overfit the resulting model. If we focus on kNN weighting methods, many proposals weighting features or instances can be found. In Raymer, Punch, Goodman, Kuhn, and Jain (2000) a weighting method to obtain an optimal set of features was provided. The set of features was selected by means of a kNN-based genetic algorithm using a bit vector to indicate if a feature was in the selection or not. In a later work, the same authors presented a hybrid evolutionary algorithm using a Bayesian discriminant function (Raymer, Doom, Kuhn, & Punch, 2003) and trying to isolate characteristics belonging to large datasets of biomedical origin. Moreover, Paredes and Vidal (2006) used different similarity functions to improve the behaviour of the kNN. In a first approximation, they considered a weight by feature and instance on training data resulting in a non-viable number of parameters in the learning process. Then, the authors proposed three types of reduction: a weight by class and feature (label dependency), a weight by prototype (prototype dependency) and a combination of the previous ones. The optimization process was carried out by descendant gradient. In the same line, Tahir, Bouridane, and Kurugollu (2007) showed an approach that was able to both select and weight features simultaneously by using tabu search. Furthermore, Mohemmed and Zhang (2008) presented a nearest-centroid-based classifier. This method calculated prototypical instances by considering arithmetic average from the training data. To classify an instance, the method calculated the distance to every prototype and then selected the nearest one. The optimization of the best centroids that minimized the classification error was carried out through particle swarm. Fernandez and Isasi (2008) also proposed a weighting system by using a prototype-based classifier. After a data normalization that was based on the position of the

* Correspoding author. Tel.: +34954555964, Fax.: +34954557139.
E-mail addresses: mateosg@us.es (D. Mateos-García), jorgarcia@us.es (J. García-Gutiérrez), riquelme@us.es (J.C. Riquelme-Santos).
URL: http://www.lsi.us.es (D. Mateos-García)

http://dx.doi.org/10.1016/j.eswa.2015.08.017
0957-4174/© 2015 Elsevier Ltd. All rights reserved.

instances with respect to regions, the weights were iteratively calculated. More recently, AlSukker, Khushaba, and Al-Ani (2010) used differential evolution to find weights for different aspects of data. They described four approaches: feature weighting, neighbour weighting, class weighting and mixed weighting (features and classes), with the latter being the one providing the best results.

Weighting has also been applied to the vote system of the kNN. Thus, the distance-weighted k-nearest neighbour rule (WKNN) proposed by Dudani (1976) has been known for long. WKNN weights the votes of the k nearest neighbours (w_i) according to Eq. (1) where d_i is the distance of the ith nearest neighbour (being d_1 the distance of the nearest) regarding an instance to be classified. A similar version using a uniform weighting (UWKNN) has also been proposed where a weight is inversely proportional to the position reach among the neighbours (i.e., $w_i^u = 1/i$). Recently, both techniques have been explored working together as a new classifier called Dual-Weighted kNN (DWKNN) showing promising results where each weight was calculated according to Eq. (2) (Gou, Xiong, & Kuang, 2011). A later work of Gou, Du, Zhang, and Xiong (2012) provided another version of DWKNN where the calculation of the weights were different according to Eq. (3).

$$w_i^w = \begin{cases} \dfrac{(d_k - d_i)}{(d_k - d_1)} & \text{if } d_i \neq d_1 \\ 1 & \text{if } d_i = d_1 \end{cases} \tag{1}$$

$$w_i^{dw1} = w_i^w * w_i^u \tag{2}$$

$$w_i^{dw2} = \begin{cases} \dfrac{(d_k - d_i)}{(d_k - d_1)} * \dfrac{(d_k + d_1)}{(d_k + d_i)} & \text{if } d_i \neq d_1 \\ 1 & \text{if } d_i = d_1 \end{cases} \tag{3}$$

Although all the previous approaches provided improvements regarding the classical kNN performance, they have not explored the possible better suitability of evolutionary computation for the optimization of the neighbours weights. Thus, we propose an evolutionary method to improve the kNN rule by altering the vote system knowledge obtained in the training phase. We also explore the use of constraints in learning weights with two different versions of our approach and we study their reliability compared with classical kNN and other 4 weighted variants on UCI datasets (Lichman, 2013). Finally, results are statistically validated to reinforce the final conclusions.

The rest of the paper is organized as follows. Section 2 presents the elements of the two versions of the evolutionary algorithm designed to weight the vote system of the kNN. The results and several statistical tests are specified in Section 3. Finally, Section 4 presents the main findings and future work.

2. Method

In this section, two variants of our voting optimization system called *Evolutionary Voting of Nearest Neighbours* (EvoNN) are described.

2.1. Purpose and functionality

The aim of our work was to find a set of weights to modify the influence of every nearest neighbour when they voted to assign a label to an unlabelled instance. Moreover, our approach also provided a measurement about the influence of the proximity of neighbours by means of the optimized weights. Thus, the evolutionary process provides the optimal weights that have been found to improve the classification accuracy of the domain under study.

To formalize our approach we assume that the set of classes (or labels) L is represented by the natural numbers from 1 to b, with b being the number of labels. Thus, let $D = \{(e, l) \mid e \in \mathbb{R}^f \text{ and } l \in L = \{1, 2, \ldots, b\}\}$ be the dataset under study with f being the number of features and b the number of labels. Let *label* be a function that assigns to every element e the real class to which it belongs to. Let us suppose that D is divided in the sets TR and TS, each of them being the training set and the testing set, respectively, such that $D = TR \cup TS$ and $TR \cap TS = \emptyset$. In the training step the classification error is minimized with participation only by instances from TR. This classification error is calculated as follows. For each $x \in TR$ we compute its k nearest neighbours according to a distance function d. Let x_i with $i = 1 \ldots k$ be the neighbours to x but ranked by distance, i.e.: $d(x, x_1) \leq d(x, x_2) \ldots \leq d(x, x_k)$ and $\forall y \in TR$ with $y \neq x_i$, $d(x, x_k) < d(x, y)$. According to the standard kNN rule, the prediction of the label of x from the labels of its neighbours can be formalized:

$$predLabel(x) = \arg \max_{l \in \{1..b\}} \sum_{i=1}^{k} \delta(l, label(x_i)) \tag{4}$$

where

$$\delta(l, label(x_i)) = \begin{cases} 1 \text{ if } label(x_i) = l \\ 0 \text{ otherwise} \end{cases} \tag{5}$$

If instead of a unitary vote, we consider that the ith neighbour contributes with the weight $w_i \in \mathbb{R}$, the function (4) is redefined:

$$predLabel(x, w) = \arg \max_{l \in \{1..b\}} \sum_{i=1}^{k} \omega_i \delta(l, label(x_i)) \tag{6}$$

The function to minimize is the sum of all prediction errors of every instance x from TR. Thus, if we define the error function as

$$error(x, w) = \begin{cases} 1 \text{ if } predLabel(x, w) \neq label(x) \\ 0 \text{ otherwise} \end{cases} \tag{7}$$

then the function to optimize is:

$$\min_{w \in \mathbb{R}^k} \sum_{x \in TR} error(x, w) \tag{8}$$

With regard to the two versions of the evolutionary algorithm, they were called EvoNN1 and EvoNN2. For EvoNN1, the nearest neighbours (those with lowest distances regarding the unlabelled instance) were " heavier" and therefore, their influence (weight) had to be greater. In EvoNN2 weights had no constraints. Regarding the design of the evolutionary algorithm, it is easy to suppose that EvoNN1 and EvoNN2 were similar except on the encoding, crossover and mutation.

2.2. Voting optimization

This subsection details the search algorithm to calculate the optimum contribution of k nearest neighbours carried out by two versions of an evolutionary algorithm . It is then necessary to define their main characteristics i.e., individual encoding, genetic operators, fitness function and generational replacement policy.

2.2.1. Individual encoding

In EvoNN1 and EvoNN2, an individual was a real-valued vector with size k symbolizing the relative contribution of the k-nearest neighbours in the voting system of the kNN rule. Position 1 in the vector was associated with the nearest neighbour in distance and position k with the furthest. Moreover, a constraint was established in EvoNN1 to assure that the closest neighbours were more important i.e., $\omega_1 \geq \omega_2 \geq \ldots \omega_k$.

Regarding the initial population, both designs integrated individuals with k random values between 0 and 1 (ordered in EvoNN1). To include individuals representing classic kNN, initial population also included several vectors with the first k values set to 1 and the remaining set to 0 in the initial population e.g., $(1.0, 0.0, \ldots, 0.0)$ for