Applied Soft Computing 40 (2016) 569–580

Contents lists available at ScienceDirect

Applied Soft Computing

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

PGGP: Prototype Generation via Genetic Programming

Hugo Jair Escalante[a,*], Mario Graff[b], Alicia Morales-Reyes[a]

[a] Computer Science Department, Instituto Nacional de Astrofísica, Óptica y Electrónica, Luis Enrique Erro 1, Puebla 72840, Mexico
[b] INFOTEC – Catedras CONACYT Centro de Investigación e Innovación en Tecnologías de la Información y Comunicación, Cto. Tecnopolo Sur 112, Ags. 20313, Mexico

ARTICLE INFO

Article history:
Received 11 July 2014
Received in revised form
12 December 2014
Accepted 11 December 2015
Available online 23 December 2015

MSC:
68T10
68T20

Keywords:
Prototype generation
Genetic programming
1NN classification
Pattern classification

ABSTRACT

Prototype generation (PG) methods aim to find a subset of instances taken from a large training data set, in such a way that classification performance (commonly, using a 1NN classifier) when using prototypes is equal or better than that obtained when using the original training set. Several PG methods have been proposed so far, most of them consider a small subset of training instances as initial prototypes and modify them trying to maximize the classification performance on the whole training set. Although some of these methods have obtained acceptable results, training instances may be under-exploited, because most of the times they are only used to guide the search process. This paper introduces a PG method based on genetic programming in which many training samples are combined through arithmetic operators to build highly effective prototypes. The genetic program aims to generate prototypes that maximize an estimate of the generalization performance of an 1NN classifier. Experimental results are reported on benchmark data to assess PG methods. Several aspects of the genetic program are evaluated and compared to many alternative PG methods. The empirical assessment shows the effectiveness of the proposed approach outperforming most of the state of the art PG techniques when using both small and large data sets. Better results were obtained for data sets with numeric attributes only, although the performance of the proposed technique on mixed data was very competitive as well.

1. Introduction

Pattern classification is the task of associating objects with labels, where objects are usually represented by numerical vectors. The field has been studied extensively and a wide diversity of methods are available out there (see e.g., [13]). Among the most popular pattern classification methods are those based on similarity or distance estimation. This type of methods rely on similarity measures to assign labels to new objects, a representative classifier of this methodology is KNN [3]. Similarity-based methods have proved to be very effective on classical pattern classification tasks, including handwritten digit recognition and text categorization. However, despite their acceptable performance, they require computing similarity estimates with all of the training objects when a new instance needs to be classified, which can be computationally expensive; besides, this type of methods require considerable storage resources and they can be sensitive to noisy instances.

Prototype-based classifiers aim at alleviating the above issues by using only a subset of representative instances for classification instead of the whole training set. The main goal of prototype-based classifiers is to achieve comparable performance to methods that use the whole data set of instances, while reducing the computational cost and storage requirements. The key issue in prototype-based classification is that of determining what are the prototypes to be used for classification. There are two main alternatives for solving this problem: selection and generation of prototypes. In the former approach, a subset of the whole set of training objects is selected as the set of prototypes [20,10]. The second approach consists of generating a set of representative instances by using information from the data set of objects [23]. Although there are not comprehensive studies comparing generation and selection strategies,[1] generation methods are more general than selection ones and, in fact, prototype selection can be considered a special case of prototype generation (PG) [23].

This paper introduces a genetic programming approach to the PG problem. The proposed method combines instances from the original training set to produce prototypes. The instances to be merged and the combination strategy are automatically determined via genetic programming, where the genetic program aims at generating prototypes that maximize an estimate of the generalization performance of an 1NN classification rule. The proposed strategy automatically selects the number of prototypes per class. Also, it generates prototypes by combining many training examples, in contrast to previous works that consider training instances only to guide the search process. Additionally, the formulation of the problem allows us to generate prototypes that are non-linear combinations of instances, which may help us to better characterize the original input space.

An experimental assessment of the proposed strategy is carried out using a suite of benchmark pattern-classification problems [23]. The considered benchmark allows us to compare the performance between the proposed approach and the most representative PG methods. In terms of accuracy and data set reduction, the proposed method outperforms most alternative approaches when considering both small and large data sets. Despite its effectiveness, the intuitive idea behind the

* Corresponding author. Tel.: +52 222 2663100x8319; fax: +52 222 2663152.
E-mail addresses: hugojair@inaoep.mx (H.J. Escalante), mgraffg@gmail.com (M. Graff), a.morales@inaoep.mx (A. Morales-Reyes).

[1] One should note that there are works comparing a few selection and generation methods over a small number of data sets [17,14].

http://dx.doi.org/10.1016/j.asoc.2015.12.015
1568-4946/© 2015 Elsevier B.V. All rights reserved.

proposed method is very simple and there are many ways in which this approach can be extended.

One should note that although our main goal is to generate prototypes for pattern classification with KNN techniques, there are other tasks and problems that could be benefit by PG techniques like ours, including: oversampling, where the goal is generating new/artificial instances for reducing the class-imbalance problem; codebook learning, where one wants to learn a set of instances that can be used as reference to represent more complex objects (e.g., images [5] or videos [25]); instance-weighting for domain adaptation, where one wants to find important instances in the source domain to be exploited in the target domain, among other tasks. Besides, it has been shown very recently that a similar formulation can be adopted to generate features [11].

The rest of this paper is organized as follows. Next section reviews related work on PG emphasizing those methods based on heuristic optimization. Section 3 introduces the proposed GP approach. Section 4 describes experimental settings and reports experimental results obtained by the proposed strategy. Finally, Section 5 outlines conclusions and future work directions.

2. Related work

Triguero et al. [23] presented a taxonomy and a comparative study among several PG methods.[2] A total of 32 different strategies are classified and an experimental comparison among 25 of these methods is reported. In that study, the method achieving the highest classification performance is GENN (Generalized Editing using NN) [15], a detrimental method that removes and relabels instances. GENN is a conservative method because it aims to edit only to an extent that does not harm significantly the classification performance, it achieves substantially better results than any other of the PG methods compared in [23]; although it was among the worse in terms of reduction. On the other hand, PSCSA (Prototype Selection Clonal Selection Algorithm) obtained the best performance in terms of reduction [9]. PSCSA models the PG problem with an artificial immune system: the clonal selection algorithm. This method is able to exactly select a single example per class, achieving the best reduction performance among the other 24 methods considered in [23]. However, its performance in terms of accuracy is worse in comparison to several other strategies. Hence, a lesson learned from [23] is that a good PG method must observe an acceptable tradeoff between accuracy and reduction.

PG methods proposed so far are very diverse, in recent studies PG methods based on bio-inspired optimization have reported better results than alternative approaches [23,24,9,8,19,1]. Usually, these methods start from a set of solutions (sets of prototypes) and modify them according to specific operators, through an iterative search procedure that attempts to optimize a criterion related to the prototypes classification performance.

A PG method based on particle swarm optimization (PSO) was proposed in [19]. A standard PSO algorithm was designed to attempt to minimize the classification error in the training set. The method is run for several times in order to obtain varied solutions. When classifying a new object the outputs of all prototypes set are combined via voting. The ensemble strategy allows this method to obtain better results than many other methods evaluated in [23]. Another variant of PSO, adaptive Michigan PSO (AMPSO), has also been used for generating prototypes. In AMPSO each swarm particle is associated to a prototype in such a way that the whole population is the set of prototypes to optimize [1].

Regarding evolutionary algorithms, successful approaches have been proposed as well. For instance, ENPC (Evolutionary Design of NN classifiers) proposed in Fernandez and Isasi [8] is an evolutionary algorithm that starts from a single individual that is evolved by applying a variety of operators to combine and split prototypes. This method is able to automatically determine the number of prototypes and requires little information from the user. ENPC is able to obtain competitive performance in terms of accuracy but it is not among the best methods in terms of reduction [23].

In most of the above reviewed PG methods, only a small subset (the initial prototypes) of training instances has an impact into the resultant set of prototypes; even when the whole training set is considered to assess the prototypes quality. It is important to notice that important information could be captured if more training samples are considered to build prototypes. The PG method proposed in this research aims to amend this issue by generating prototypes from combinations of training instances, where any training instance can be considered during the optimization process. Additionally, the way we formulate the problem allows us to obtain prototypes that are non-linear combinations of training instances, which may be helpful to better characterize the original input space. In Section 4 the proposed GP method demonstrates a better tradeoff than most of the methods reviewed in this section and others considered in Triguero et al. [23].

The proposed PG method is based on genetic programming, an evolutionary algorithm where individuals can be complex data structures [21]. Genetic programming has a long history in machine learning and pattern classification [16]. However the most common application for genetic programming is related to the generation of either decision-trees or rule-induction based classifiers [7,16]. To the best of our knowledge no PG method based on genetic programming has been proposed so far. In Cordella et al. [2], authors describe a genetic program for producing what they call "*classification prototypes*". However, those "prototypes" are in fact logical rules used for classification, which are learned via genetic programming.

A preliminary version of the proposed method was reported in [6], this paper extends that publication by providing a more comprehensive and detailed description of the proposed method. Moreover, the previous algorithmic approach has been improved by removing a genetic operator and extensive experimental results are reported to evaluate the performance of the proposed strategy while considering different settings.

3. PGGP: Prototype Generation via Genetic Programming

This paper introduces PGGP: a method for Prototype Generation via Genetic Programming. PGGP automatically combines instances from a particular class to generate classification prototypes for that class. The combination strategy is determined by a genetic program that aims at maximizing an estimate of the generalization performance of an 1NN classifier. Although the prototypes of a class are determined only by examples of its class, under the proposed approach the prototypes for all classes are dependent on each other, because the fitness function evaluates the set of prototypes as a whole; hence making prototypes suitable for discrimination. PGGP selects appropriate instances for combination and, at the same time, determines which operators should apply in order to combine these instances and generate prototypes. Moreover, since in most classification problems is unrealistic to know beforehand the number of prototypes, the genetic program is freely allowed to learn the best number of prototypes for each class. The rest of this section describes in detail the proposed PGGP method.

3.1. Considered scenario

Let $\mathcal{T} = \{(\mathbf{x}_1, y_1) \ldots, (\mathbf{x}_N, y_N)\}$ be a training set of labeled instances, with $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \mathcal{C} = \{1, \ldots, K\}$, where d is the problem dimensionality and K is the number of classes in the considered problem. The goal is to generate a set of prototypes $\mathcal{P} = \{(\mathbf{w}_1, y_1) \ldots, (\mathbf{w}_L, y_L)\}$, such that $L \ll N$, where $\mathbf{w}_i \in \mathbb{R}^d$ and there is at least one prototype associated to each class in $\mathcal{C}$. The considered scenario is graphically described in Fig. 1.

[2] See also: http://sci2s.ugr.es/pr/.