Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Systems and Software 76 (2005) 277–285
The Journal of Systems and Software
www.elsevier.com/locate/jss

Genetic granular classifiers in modeling software quality

Witold Pedrycz [a,b,*], Giancarlo Succi [c]

[a] Department of Electrical and Computer Engineering, University of Alberta, Edmonton, AB, Canada T6G 2G7
[b] Systems Research Institute of Polish Academy of Sciences, Warsaw, Poland
[c] Department of Computer Science, University of Bozen, I-39100 Bozen, Italy

Received 4 July 2003; received in revised form 9 June 2004; accepted 12 June 2004
Available online 26 August 2004

Abstract

Hyperbox classifiers are one of the most appealing and intuitively transparent classification schemes. As the name itself stipulates, these classifiers are based on a collection of hyperboxes—generic and highly interpretable geometric descriptors of data belonging to a given class. The hyperboxes translate into conditional statements (rules) of the form "if feature$_1$ is in $[a,b]$ and feature$_2$ is in $[d,f]$ and ... and feature$_n$ is in $[w,z]$ then class ω" where the intervals $([a,b],\ldots,[w,z])$ are the respective edges of the hyperbox. The proposed design process of hyperboxes comprises of two main phases. In the first phase, a collection of "seeds" of the hyperboxes is formed through data clustering (realized by means of the Fuzzy C-Means algorithm, FCM). In the second phase, the hyperboxes are "grown" (expanded) by applying mechanisms of genetic optimization (and genetic algorithm, in particular). We reveal how the underlying geometry of the hyperboxes supports an immediate interpretation of software data concerning software maintenance and dealing with rules describing a number of changes made to software modules and their linkages with various software measures (such as size of code, McCabe cyclomatic complexity, number of comments, number of characters, etc.).

Keywords: Software quality; Hyperbox geometry of classifiers; Software measures; Genetic algorithms; Fuzzy clustering; Fuzzy C-Means (FCM)

1. Introductory comments

Pattern classifiers (Duda and Hart, 1973) come with their underlying geometry. As a matter of fact, the geometry and related learning mechanisms are the two essential elements that determine the resulting performance of any pattern classifier. What we witness today is an evident panoply of technologies used to design classifiers: neural networks, fuzzy rule-based systems (and ensuing fuzzy classifiers), k-Nearest Neighbor (k-NN) classifiers, and genetic optimization are just a few representative categories of pattern classifiers. The assumed geometry of the classifier (say, hyperplanes, receptive fields, etc.) implies its learning capabilities and resulting accuracy. Quantitative Software Engineering inherently dwells on empirical data that need to be carefully analyzed. As there are no physical underpinnings characterizing software processes and ensuing software products, the commonly encountered assumptions that govern "standard" regression models and classifiers do not hold and could be difficult to justify. On the other hand, we anticipate that models developed in the realm of Software Engineering should be transparent meaning that their readability and a level of comprehension are high. Being faced by the lack of physical underpinnings on one hand and the need for the user-friendliness on the other, it becomes advisable to set up a logic-based development framework and adopt logic as a paramount feature of the resulting constructs. In this way the interpretability of the models is inherently associated with the logic roots of the environment. Likewise their geometry is also implied by the logic fundamentals we have started with at the beginning.

* Corresponding author. Tel.: +1 780 492 4661; fax: +1 780 492 1811.
E-mail address: pedrycz@ee.ualberta.ca (W. Pedrycz).

0164-1212/$ - see front matter © 2004 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2004.06.018

The quest for interpretability of models in Software Engineering comes with a variety of facets and diverse applications including software specification, software maintenance, reliability, and portability. Apparently, some categories of models are geared towards interpretational clarity and make it one of the dominant features. This becomes visible in rule-based systems, cf. (De Falco et al., 2002; Gabrys and Bargiela, 2001; Pedrycz et al., 2002) in which this feature comes hand in hand with the requirement of high accuracy and substantial prediction capabilities. Overall, there are several general observations worth making with this respect

- software processes and products are human-centric and do not adhere to any physical underpinnings. It light of their origin, it is legitimate to focus on logic-rich and transparent models,
- domain knowledge becomes an integral part of the model especially in case of its availability and limited availability of experimental data as well as substantial variability of the software products and processes,
- interpretability of the developed models becomes an important and highly desirable feature of models of software artifacts which helps designer and manager gain a better insight into the specificity of the particular model and derive conclusions. In a nutshell, the geometry of the model (say a predictor or classifier) needs to be easily comprehended by the user,
- software measures (metrics) (Garmus and Herron, 1996; Muller and Paulish, 1993; Munson and Khoshgoftaar, 1996; Pedrycz et al., 2001) become essential indicators of software quality (such as reliability, maintenance effort, development cost, etc.). It becomes then essential to develop models that are easily understood by the managing personnel and designers and help look at possible scenarios and pursue any detailed "what-if" considerations.

The geometry of the feature space imposed by the classifier is also inherently associated with our ability to interpret classification processes and comprehend the decision boundaries produced by the classifier. In general, these are nonlinear. In an ideal situation they may coincide with those produced by a Bayesian classifier (Duda and Hart, 1973). From the interpretation point of view, the most intuitive ones are those built in the form of boxes (in two-dimensional space) or hyperboxes (in multi-dimensional space), refer to Fig. 1.

Subsequently, when using boxes (and hyperboxes in general), any classification rule becomes straightforward and emerges as a result of enlisting of the edges of the boxes:

assign (classify) pattern $\boldsymbol{x} \in \boldsymbol{R}^n$ to class ω_1 if $\boldsymbol{x}$ belongs to one of the hyperboxes $\boldsymbol{H}_1, \boldsymbol{H}_2, \ldots, \boldsymbol{H}_c$ describing (localizing) patterns belonging to the class under interest.

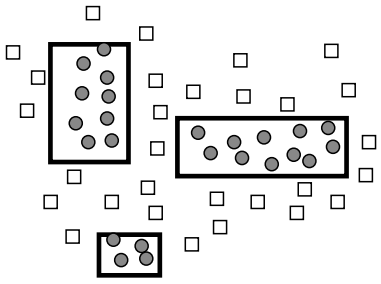

Fig. 1. Examples of hyperbox-based classifier formed in a two-dimensional space; note a number of hyperboxes (boxes) formed there and "covering" patterns (data) belonging to class ω_1 (dots). The second class is denoted by small squares and its elements are excluded from the hyperbox.

The hyperbox classifier exhibits two interesting properties. First, the concept of the classifier is profoundly simple. Potentially we can improve the classification rate by moving to the higher level of detail and increasing the number of the boxes while making them smaller and in this way refining the classifier. Second, the classifier directly translates into a set if transparent rules (since each box is a rule itself) whose condition parts assume a straightforward interpretation. Note that a hyperbox is just a Cartesian product of the intervals forming its edges. The rules read as "if $\mathbf{x}$ is in $\mathbf{H}_i$ then ω_1", $i = 1, 2, \ldots, c$. Alternatively, we can allude to the edges of the hyperbox and spell out a collection of the conditions, that is "if x_1 is in $[a_1, b_1]$ and x_2 is in $[a_2, b_2]$ and ... and $[a_n, b_n]$ then ω_i". Owing to their interpretability, hyperbox classifiers have been studied in the literature quite intensively, cf. (Gabrys and Bargiela, 2001; Simpson, 1992; Simpson, 1993; Pedrycz et al., 2002). The most popular approach to the design of these classifiers is perhaps the one proposed by Simpson (1992, 1993) where he discusses both supervised and unsupervised mode of learning. What somewhat hampers the development of the hyperbox classifiers is a lack of learning algorithms which is not so surprising considering the geometry of the classifiers (that do not come with differentiable boundaries of the hyperboxes and in this way are not suitable for gradient-based optimization techniques).

The objective of this study is to develop a hybrid, two-phase design of hyperbox classifiers and discuss their essential role in analysis and classification of software data. In the first phase of the design, we "seed" the hyperboxes by using fuzzy clustering in which designed are the prototypes (centers) of the clusters. In our case there play a role of seeds around which the we start "growing" the boxes by expanding the size of the box and pushing its walls further from the center. This process is followed by the second phase in which we "grow" the hyperboxes via genetic optimization. This hybrid approach helps us capture the nature of