Extraction of prototype-based threshold rules using neural training procedure

Marcin Blachnik[1], Mirosław Kordos[2], Włodzisław Duch[3]

[1] Silesian University of Technology, Department of Management and Informatics, Katowice, Krasinskiego 8, Poland: marcin.blachnik@polsl.pl
[2] University of Bielsko-Biala, Department of Mathematics and Computer Science, Bielsko-Biała, Willowa 2, Poland: mkordos@ath.bielsko.pl
[3] Nicolaus Copernicus University, Department of Informatics, Grudziądzka 5, Toruń, Poland, and School of Computer Science, Nanyang Technological University, Singapore;

Abstract. Complex neural and machine learning algorithms usually lack comprehensibility. Combination of sequential covering with prototypes based on threshold neurons leads to a prototype-threshold based rule system. This kind of knowledge representation can be quite efficient, providing solutions to many classification problems with a single rule.

Keywords: Data understanding, rule extraction, prototype-based rules

1 Introduction

Neural networks and other complex machine learning models usually lack the advantages of comprehensibility. This property is very important in many application, including safety-critical ones. Also in technological applications, where the systems are used for support of industrial processes (see for example [**?**]) simple and understandable models are crucial to avoid dangerous situations and to raise process engineers' confidence in the solutions. The second aspect of building a comprehensible model is directly related to knowledge extraction. In medical applications or social science the data driven models may be the only source of knowledge about certain processes. Thus there are evident advantages of data driven models, which use human-friendly knowledge representation.

Four general approaches, which find comprehensive mappings $f(\mathbf{x}_i) \rightarrow y_i$ are usually considered in that purpose [**?**]:

- propositional logic using crisp logic rules (C-rules)
- fuzzy logic and fuzzy rule based systems (F-rules)
- prototype-based rules and logic (P-rules)
- first and higher-order predicate logic

C-rules are the most common form of user-friendly knowledge representation. They avoid any ambiguity, which assures that there is only one possible interpretation of

the rules, although at the expense of several limitations. Continuous attributes usually cannot be discretized in a natural way, crisp decision borders divide the input space into hyperboxes that cannot easily cover some data distributions, and some simple forms of knowledge (like "the majority vote") may be expressed only using a very large number of logical conditions in propositional form. The problems with discretization have been addressed by fuzzy logic, with fuzzy rule-based systems using membership functions to represent the degree (in $[0, 1]$ range) of fulfilling certain conditions [**?**]. This leads to more flexible decision borders and allows for handling uncertainty in the real world data. However, F-rules do not represent many forms of knowledge that predicate logic can express in a natural way.

Prototype based rules try to capture intuitive estimation of similarity in decision-making processes [**?**], as it is done in the case-based reasoning. P-rules, which are represented by reference vectors and similarity measures are more suitable to learn from continuous data than predicate logic. Relation of P-rules with F-rules has been studied in [**?**,**?**] where it has been shown that fuzzy rules can be converted to prototype-rules with additive distance functions. P-rules work with symbolic attributes by applying heterogeneous distance measures like VDM distance [**?**]. Moreover, they can easily express some forms of knowledge, such as the selection of the *m of n* true conditions, what limits the use of crisp and fuzzy rules.

P-rules can be expressed in two forms: as the nearest neighbor rules or as the prototype-threshold based rules. This article addresses the problem of extracting the prototype-threshold based rules from the data. A simple algorithm called *nOPTDL* based on combination of the sequential covering approach to rule extraction with neural-like training and representation of single rules is presented below. it allows to use gradient-based optimization methods to optimize the parameters of neurons, which represent single rules.

In the next section the details of prototype-threshold rules are discussed and in section 3 the *nOPTDL* algorithm is presented. Section 4 presents a few examples of the *nOPTDL* performance and discusses the results. The last section concludes the paper and presents the directions of further research.

In the paper the notation, which describes the data is given as a tuple $\mathbb{T} = \{[\mathbf{x}_1, y_1], [\mathbf{x}_2, y_2], \ldots, [\mathbf{x}_n, y_n]\}$, where $\mathbf{x}_i \in \mathbb{R}^m$ is an input vector, $y_i \in [-1, 1]$ is a label (only two-class problems are considered).

2 Prototype-threshold based rules

A single prototype-threshold based rule has the form:

$$\text{If } D(\mathbf{x}_i, \mathbf{p}) < \theta \text{ Then } y_i \leftarrow l \tag{1}$$

where $D(\mathbf{x}_i, \mathbf{p})$ expresses the distance between vector $\mathbf{x}_i$ and the prototype $\mathbf{p}$ and l is the class label associated with the rule.

There are several approaches to construct this type of rules. Perhaps the simplest one is based on classical decision trees. First, conditions that define each branch of the tree are used to define a separate distance function for a single prototype associated with the root of the tree. A more common approach leading to a lower number of rules starts

from a distance matrix $D(\mathbf{q}, \mathbf{w})$ that is used to construct new features for the decision tree training. Each new feature represents the distance from a selected training vector, so the number of new features added to the original set is equal to the number of training instances $m = n$. In this approach each node can consist either of a single prototype-threshold rule or a combination of crisp conditions with distance-based conditions [**?**].

Another approach called *ordered prototype-threshold decision list OPTDL* is derived from the sequential covering principle. In this approach, described in [**?**], the algorithm starts from creating a single rule and then adds new rules such that each new rule covers examples not classified by previously constructed rules (sketch (1)). The shape of the decision border of a single rule depends on the distance function. Euclidean distance function creates hyperspherical borders. To avoid unclassified regions the new rules should overlap with each other. When a test vector falls in such an overlapping region a unique decision is made by ordering the rules from the most general to the most specific. The training algorithm starts from creating the most general rule and then each new rule is marked as more specific then the previous one. The decision-making process starts from analyzing the most specific rule and if its conditions are not fulfilled then more general rules are being analyzed. If an instance is not covered by any rule then the *else* clause is used to determine the default class label.

Algorithm 1 Sequential covering algorithm

Require: $\mathbb{T}, minSupport, maxIterations$
  $\mathbb{P} \leftarrow \emptyset$ {set of prototype rules}
  $\mathbb{S} \leftarrow \mathbb{T}$ {set of uncovered or misclassified examples}
  $i \leftarrow 0$
  repeat
    $\{\mathbf{p}_i, \theta_i\} \leftarrow CreateNewRule(\mathbb{S}, \mathbb{T})$
    $\mathbb{P} \leftarrow \mathbb{P} \cup \{\mathbf{p}_i, \theta_i\}$
    $else \leftarrow DetermineElseCondition(\mathbb{S})$
    $\mathbb{S} \leftarrow ApplyRules(\mathbb{T}, \mathbb{P})$
    $\mathbb{S}' \leftarrow ApplyElse(\mathbb{S}, else)$
    $i \leftarrow i + 1$
  until $(|\mathbb{S}'| < minSupport)$ or $(i \geq maxIterations)$
  return $\mathbb{P}, else$

Construction of a single rule requires determination of the prototype position and its corresponding threshold, what is performed based on search strategies and criteria commonly used in decision trees *sOPTDL*, such as the *information gain*, *Gini index* or *SSV* separability measure [**?**]. The algorithm considers each input vector $\mathbf{x}_i$ as a potential prototype and the best tuple $[\mathbf{p}, \theta, l]$, where θ is a threshold and l is the consequence of the rule, is optimized using search strategy sketched in Algorithm (2), where $\mathbb{S}$ is the set of examples that are either unclassified or incorrectly classified.

In experiments reported in [**?**] this approach worked quite well, although the positions of all prototypes were restricted to one of the instances of the training set. That makes the interpretation of resulting rules easy because a prototype is a real example from the training set. On the other hand that reduces the ability to create an arbitrary

Algorithm 2 Search based ordered prototype-threshold decision list algorithm (*sOPTDL*)

Require: $\mathbb{S}, \mathbb{T}$
  for $i \in \mathbb{T}$ **do**
    $l \leftarrow y_i$
    for $k \in |\mathbb{S}|$ such that $y_{k-1} \neq y_k$ **do**
      $\theta_k = 0.5 \cdot (d(\mathbf{x}_i, \mathbf{x}_k) - d(\mathbf{x}_i, \mathbf{x}_{k-1}))$
      $v \leftarrow Criterion(\mathbf{x}_i, \theta_k, \mathbb{T}_{y \neq l}, \mathbb{S}_{y=l})$
      if $v > v'$ **then**
        $v' \leftarrow v$
        $\theta' \leftarrow \theta_k$
        $\mathbf{p} \leftarrow \mathbf{x}_i$
      end if
    end for
  end for
  return $\mathbf{p}, \theta'$

shape of the decision border. For two overlapping Gaussian distributions with identical σ the optimal decision border is a hyperplane, but such decision border cannot be created using prototypes restricted to the examples of the training set. Without that restriction a prototype can be moved to infinity and with appropriately large threshold a good approximation to a linear decision border can be obtained. In the next section presents the optimization procedures to determine the position and appropriate threshold of a prototype.

3 Neural Optimization of Prototype-Threshold Rules

The goal here is to determine the optimal position of the prototype and its associated threshold. This is done by optimization of parameters of the neurons that implement hyperspherical isolines, such that each coverage step of the rule induction consists of a single neuron training (the *nOPTDL* algorithm). The transfer function of that neuron is based on a modified logistic function:

$$z(\mathbf{x}|\mathbf{p}, \theta) = \sigma\left(D(\mathbf{x}, \mathbf{p})^{\alpha} - \theta\right) \tag{2}$$

$$\sigma(x) = \frac{1}{(1 + \exp(-x))} \tag{3}$$

where $\mathbf{p}$ is the position of the prototype, $D(\mathbf{x}, \mathbf{p})$ is the distance function, α represents the exponent of the distance function (for Euclidean distance $\alpha = 2$) and θ denotes the threshold or bias. The α parameter is used to add flexibility to distance functions, regulating their shape as a function of differences between vectors.

The inner part of the transfer function $g(\mathbf{x}) = D(\mathbf{x}, \mathbf{p})^{\alpha} - \theta$ defines the area covered by active neuron, such that vectors $\mathbf{x}$ that fall into this area give positive values

$g(\mathbf{x}) > 0$ and those being outside - negative values $g(\mathbf{x}) < 0$. The logistic function is used for smooth nonlinear normalization of the $g(\mathbf{x})$ values to fit them into the range $[0, 1]$. $\mathbf{x}$ vectors close to the border defined by $z(\cdot) = 0.5$ will increase this value towards 1 inside and towards 0 outside the area covered by the neuron, with the speed of change that depends on the slope of the logistic function and the scaling of the distance function.

The objective function used to optimize the properties of the neuron is defined as:

$$E(\mathbf{p}, \theta) = \sum_{i \in \mathbb{C}} z(\mathbf{x}_i | \mathbf{p}, \theta) \cdot l \cdot y_i \tag{4}$$

which is a sum of neuron activations multiplied by the product of the consequence $l = \pm 1$ of the rule associated with the prototype $\mathbf{p}$. $\mathbb{C}$ denotes a set of training examples incorrectly classified ($\mathbb{T}$ with $l \neq y_i$) and samples that are not yet covered by the current set of rules ($\mathbb{T}$ for which $l = y_i$).

The objective function can be optimized using either gradient or non-gradient optimization methods. To avoid local minima and speed up convergence gradient optimization procedure that is restarted from 5 different random localizations is used, each time starting from a vector that is not yet properly classified.

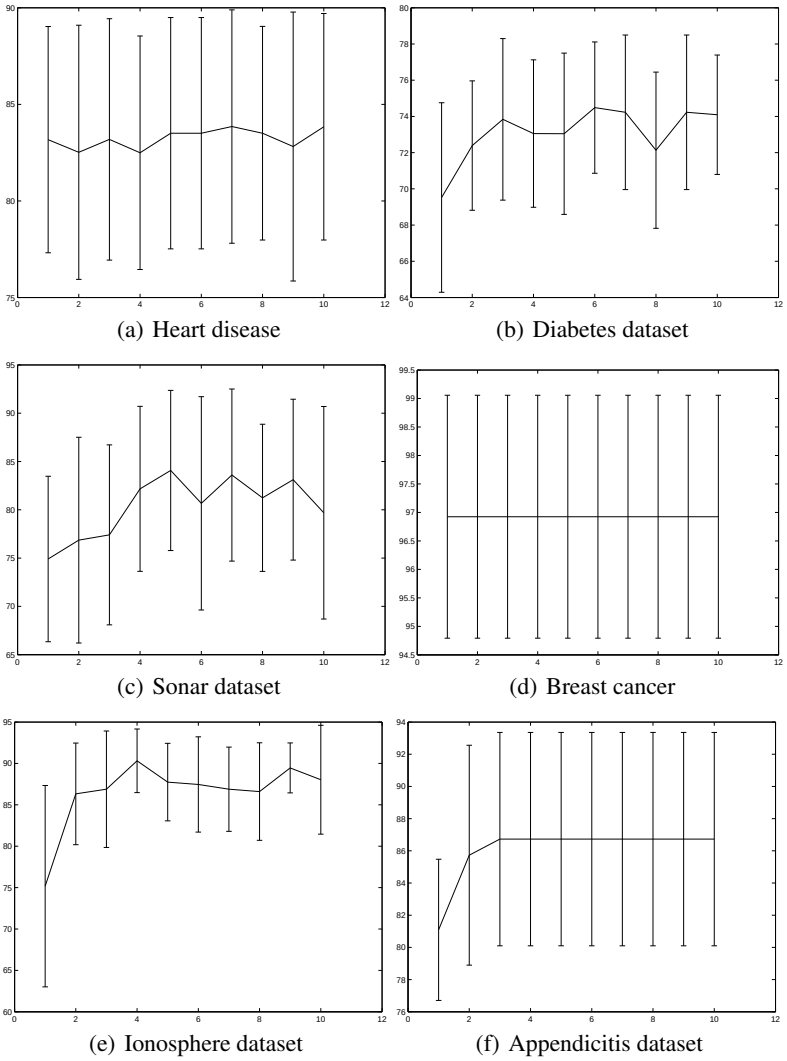
4 Numerical experiments

We experimentally compare the accuracy and comprehensibility of rules induced by our system. The experiments were performed using 6 benchmark datasets with different properties taken from the UCI repository [**?**]. They include: the Cleveland heart disease (Heart disease), Pima Indian diabetes (Diabetes), sonar (Sonar), Wisconsin breast cancer (Breast cancer) and appendicitis (Appendicitis). The properties of these datasets are summarized in Tab. (1). These datasets represent quite diverse domains, including medical data with heterogeneous type of attributes and datasets with many continuous attributes, such as the Sonar dataset, which are difficult to handle using crisp rules [**?**].

Table 1. Description of the datasets used in rule extraction experiments.

Dataset	# vectors	# features	# of classes	comment
Heart disease	297	13	2	6 vectors with missing values were removed
Diabetes	768	8	2	
Sonar	208	60	2	
Breast cancer	683	9	2	16 vectors with missing values were removed
Ionosphere	351	34	2	
Appendicitis	106	8	2	

In the first experiment the influence of the number of extracted rules on classification accuracy of the *nOPTDL* has been analyzed. A 10-fold crossvalidation has been used for estimation of accuracy, repeating the test for different number of rules in the range $k = [1 \ldots 10]$. The results are presented in Fig. (1).



(a) Heart disease (b) Diabetes dataset

(c) Sonar dataset (d) Breast cancer

(e) Ionosphere dataset (f) Appendicitis dataset

Fig. 1. Classification accuracy and variance as the function of the number of the *nOPTDL* rules.

The obtained results show that the classification accuracy using just a single P-rule is sometimes as good as with many rules (heart, breast cancer). In other cases adding new rules improves accuracy up to a certain point, but for all datasets no more than 5 rules were needed to reach the maximum accuracy. This shows that prototype-threshold form of knowledge representation can be quite efficient.

To compare the proposed *nOPTDL* algorithm to other state-of-the-art rule extraction algorithms another test was performed using double crossvalidation: the inner cross-validation was used to optimize parameters of the given classification algorithm (for

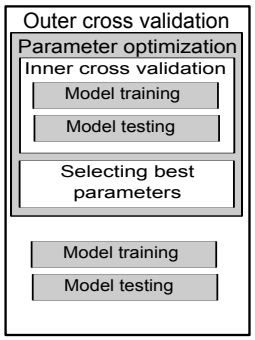


Fig. 2. The accuracy estimation procedure

example, the number of rules in our system) and the outer crossvalidation was used to predict the final accuracy. The testing procedure is presented in Fig.(4).

Our *nOPTDL* algorithm has been compared to the previous version based on search strategies (*sOPTDL*), and also to the C4.5 decision tree [**?**] and Ripper rule induction system [**?**]. The experiments have been conducted using RapidMiner [**?**] with Weka extension and with the Spider toolbox [**?**]. The parameters of both C4.5 and Ripper algorithms have also been optimized using double crossvalidation, optimizing *pureness* and *the minimal weights of instances*. The results are presented in Tab. (2).

Table 2. Comparison of the accuracy of the *nOPTDL* algorithm with C4.5 decision tree and Ripper rule induction.

Dataset	nOPTDL	sOPTDL	C4.5 [4]	Ripper [5]
	Acc±Std	Acc±Std	Acc±std	Acc±std
Heart disease	**83,5±5,76**	80.48±4.33	77,2±4,3	80,13±7,24
Diabetes	72,00±4,4	71.62±4.01	74,2±4,7	**74,61±2,66**
Sonar	**81,12±11,42**	75.02±8.91	72,5±11,2	**79,76±6,8**
Breast cancer	96,92±2,13	**96.93±1.08**	95,28±4,7	96,28±1,7
Ionosphere	88,05±5,26	**92.02±3.51**	90,33±4,7	88,61±4,2
Appendicitis	**86,72±6,63**	82.27±11.85	83,9±6	85,81±6,2

For *Heart disease* the average accuracy of *nOPTDL* (1 rule) is over 5% higher in comparison with C4.5 classifier (21 rules) and 3% higher than the Ripper algorithm (4 rules). a very good accuracy was also achieved for the *Appendicitis* dataset. The average accuracy of the *Sonar* dataset (4 rules) was also very high, however the standard deviation, comparable to that obtained from C4.5 decision tree, was much higher than the standard deviation of *Ripper*. *Diabetes* also required 3 P-rules. In other cases a single rule was sufficient. The results show that knowledge representation using a small number of P-rules is very efficient.

5 Conclusions and future research

A modification of the OPTDL algorithm (nOPTDL) for extraction of prototype-threshold based rules has been described. Neurons implementing sigmoidal functions combined

with distance-based functions represent single P-rules. Such an approach allows for efficient gradient based optimization methods for rule extraction. Moreover, the use of VDM metric and heterogeneous distance functions make it possible go apply this method to datasets consisting of symbolic or mixed types of features.

Experiments performed on diverse types of datasets showed that a good classification accuracy can be achieved with a small number of P-rules, which is the goal of any rule induction algorithm. In most cases even one single rule leads to a rather small error rate, what proves high expressive power of prototype based knowledge representation.

Further extensions of this algorithm, including beam search instead of the best first search, should improve its quality. Our future work also includes adding local feature weights to each neuron to automatically adjust feature significance. Enforcing regularization should increase the sparsity of the obtained feature weights and lead to improvement of comprehensibility by filtering useless attributes and thus simplify the extracted knowledge. Adopting appropriate distance measures and switching to the Chebyshev distance ($L_{\inf}$ norm) may allow for classical crisp rule extraction using the same OPTDL family of algorithms.

Acknowledgment The work was founded by the grant No. ATH 2/IV/GW/2011 from the University of Bielsko-Biala and by project No. 4421/B/T02/2010/38 (N516 442138) from the Polish Ministry of Science and Higher Education.

The software package is available on the web page of *The Instance Selection and Prototype Based Rules Project* at `http:\\www.prules.org`

References