Lecture Notes in Computer Science 7553, 255-262, 2012

Extraction of prototype-based threshold rules using neural training procedure

Marcin Blachnik[1], Mirosław Kordos[2]

[1] Silesian University of Technology, Department of Management and Informatics, Katowice, Krasinskiego 8, Poland: marcin.blachnik@polsl.pl
Google: M. Blachnik
[2] University of Bielsko-Biala, Department of Mathematics and Computer Science, Bielsko-Biała, Willowa 2, Poland: mkordos@ath.bielsko.pl
Google: M. Kordos

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 may be quite powerful, providing solutions to many classification problems using a single rule.

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

1 Introduction

Neural networks and other complex machine learning models usually lack the advantage of comprehensibility. This property is very important in many application, including safety-critical applications. Also in technological applications control systems trained for support of industrial processes (see for example [1]) simple and understandable models may be crucial to avoid dangerous situations and to raise confidence of process engineers in the deployed system. The second aspect of building a comprehensible model is directly related to knowledge extraction. In medical applications or social science data driven models may be the only source of knowledge of a certain process. There is a clear need of building data driven models using human friendly knowledge representation.

In a typical classification problem labeled 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 here). Four general approaches to find comprehensible mapping $f(\mathbf{x}_i) \rightarrow y_i$ are usually considered [2]:

- 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 human friendly knowledge representation. They avoid ambiguity facilitating uniqueness of interpretation 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. 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 [3]. This leads to more flexible decision borders and allows for handling uncertainty inherent in 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 [4], as it is done in case based reasoning. P-rules are represented by reference vectors and similarity measures, and are more suited to learning from data in continuous form than predicate logic. Relation of P-rules with F-rules has been studied in [5, 6] 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 [7]. Moreover, they can easily express some forms of knowledge, such as the selection of the *m of n* true conditions, that limit the use of crisp and fuzzy rules.

P-rules can be have one of the two forms, the nearest neighbor rule, and the prototype-threshold based rules. This article address the problem of extracting the prototype-threshold based rules from the data. A simple algorithm called *nOPTDL* based on combination of sequential covering approach to rule extraction with neural-like training and representation of single rules is presented below. This allows to use gradient based optimization methods to optimize properties of neurons representing single rules.

In the following 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 its performance and contains discussion of the results. The last section concludes the paper and draws perspectives on further research in this direction.

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 $D(\mathbf{x}_i, \mathbf{p})$ is a distance function 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 is based on classical decision trees. First, conditions that define each branch of the tree may be used to define separate distance function for a single prototype associated with the root of the tree. 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 training of the decision tree. Each new feature represents distance from selected training vector, so the number of new features added to the original ones is equal to the number of training instances $m = n$. In this approach each node may be a single prototype threshold rule or a combination of crisp conditions with distance-based conditions [8].

Another approach to called *ordered prototype-threshold decision list OPTDL* derives from the sequential covering principle. In this approach, described in [9], 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 distance function. Using Euclidean distance functions creates hyperspherical borders. To avoid unclassified regions new rules should overlap with each other. When a test vector falls in such overlapping region unique decisions 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 ones. 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 position of a prototype and its corresponding threshold. To this aim we have already proposed an approach [9] based on search strategies and criteria commonly used in decision trees *sOPTDL*, such as the *information gain*, *Gini index* or *SSV* separability measure. This algorithm considers each input vector $\mathbf{x}_i$ as potential prototype, and the best tuple $[\mathbf{p}, \theta, l]$, where θ is a threshold and l is the consequence of that rule, is optimized using search strategy sketched in Algorithm (2), where $\mathbb{S}$ is the set of examples that are not classified or incorrectly classified.

In experiments reported in [9] this approach worked quite well, although the positions of all prototypes were restricted to one of the instances of the training set. Such approach improves the ability to interpret resulting rules, because a prototype is a real example from the training set. On the other hand this reduces the ability to create desired 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

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 $\mathbf{p}, \theta'$
```

be created using prototypes restricted to the examples of the training set. Abandoning this restriction a prototype moved towards infinity with appropriately large threshold created good approximation to linear decision boarder. In the next section optimization procedures to determine position and appropriate threshold of a prototype is presented.

3 Neural optimization for prototype-threshold rules

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

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

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

where $\mathbf{p}$ is the position of the prototype, $D\left(\mathbf{x}, \mathbf{p}\right)$ 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 its shape as a function of differences between vectors.

The inner part of the transfer function $g\left(\mathbf{x}\right) = D\left(\mathbf{x}, \mathbf{p}\right)^{\alpha} - \theta$ defines the area covered by active neuron, such that vectors $\mathbf{x}$ that fall into this area give positive values $g\left(\mathbf{x}\right) > 0$, and those being outside negative values $g\left(\mathbf{x}\right) < 0$. The logistic function is used for smooth nonlinear normalization of the $g\left(\mathbf{x}\right)$ 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:

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

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 current set of rules ($\mathbb{T}$ for which $l = y_i$).

This objective function can be optimized using gradient or other types of algorithms. To avoid local minima and speed up convergence gradient optimization procedure that is restarted from 5 different random localization is used, each time starting from vector that is not yet properly classified.

4 Numerical experiments

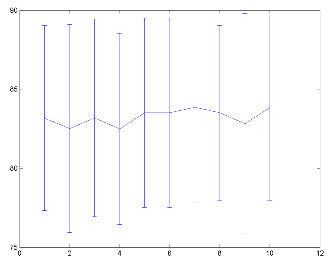
Experiments described here compare the accuracy and comprehensibility of rules induced by our system. Experiments were performed using 6 benchmark datasets with different characteristics, taken from the UCI repository [10]. They include the Cleveland heart disease (Heart disease), Pima Indian diabetes (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 applications, including medical data with heterogeneous type of attributes, and datasets with many continuous type of attributes, such as Sonar dataset, that are difficult to handle using crisp rules [2].

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

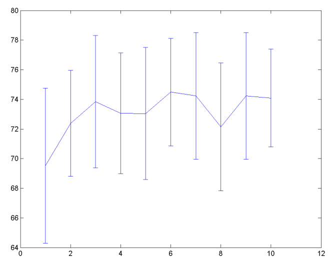
Dataset	# vectors	# features	# of classes	comment
Heart disease	297	13	2	6 vectors with missing values were removed
Diabetes	768	8	2	
Sonar	208	60	2	
Breast cancer	683	9	2	16 vectors with missing values were removed
Ionosphere	351	34	2	
Appendicitis	106	8	2	

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

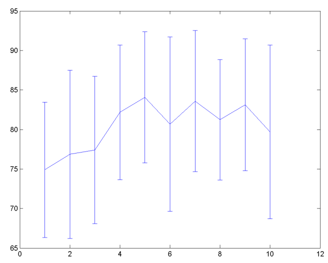
Classification accuracy using just a single P-rule is sometimes as good as with many rules (heart, breast cancer). In other cases adding new rules improves accuracy up to a point, but for all datasets no more than 5 rules were needed to reach the maximum



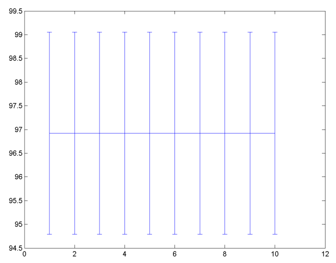
(a) Heart disease



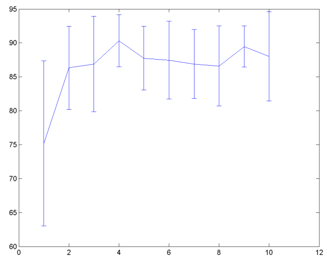
(b) Diabetes dataset



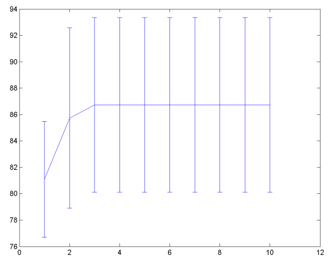
(c) Sonar dataset



(d) Breast cancer



(e) Ionosphere dataset



(f) Appendicitis dataset

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

accuracy. This shows that prototype-threshold form of knowledge representation may be quite powerful.

To compare proposed *nOPTDL* algorithm to other state-of-the-art rule extraction algorithms another test was done using double crossvalidation: the inner cross validation was used to optimize parameters of given classification algorithm (for example, the number of rules in our system), and the outer validation was used to predict the final accuracy. This testing procedure is presented in Fig.(4).

Our algorithm has been compared to the previous version based on search strategies (*sOPTDL*), and also to C4.5 decision tree [11] and Ripper rule induction system [12]. The experiments have been conducted using RapidMiner [13] with Weka extension, and with Spider toolbox [14]. The parameters of both C4.5 and Ripper algorithms has

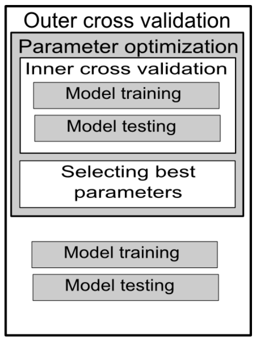


Fig. 2. The accuracy estimation procedure

also been optimized using double cross validation, optimizing *pureness* and *the minimal weights of instances*. Results are presented in Tab. (2).

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

Dataset	nOPTDL	sOPTDL	C4.5 [3]	Ripper [4]
	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 then the Ripper algorithm (4 rules). Very good accuracy was also achieved for the *Appendicitis* dataset. Average accuracy of *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* algorithm. *Diabetes* also required 3 P-rules, in other cases a single rule was sufficient. These results show that knowledge representation using small number of P-rules may be quite powerful.

5 Conclusions and future research

A simple modification of 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 approach enables efficient gradient based optimization methods for rule extraction. Moreover, the use of VDM metric and heterogeneous distance functions allows for applications of this method to datasets consisting of symbolic or mixed types of features.

Experiments performed on diverse types of datasets showed that good classification accuracy may 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, showing high expressive powers 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 importance. Enforcing regularization should increase the sparsity of obtained feature weights and lead to improvement of comprehensibility by filtering useless attributes and simplifying 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 sponsored by the Polish Ministry of Science and Higher Education, project No. 4421/B/T02/2010/38 (N516 442138).
The software package is available on the web page of *The Instance Selection and Prototype Based Rules Project* at `http:\\www.prules.org`

References

1. Wieczorek, T.: Neural modeling of technological processes. Silesian University of Technology (2008)
2. Duch, W., Setiono, R., Zurada, J.: Computational intelligence methods for understanding of data. Proceedings of the IEEE **92** (2004) 771–805
3. Nauck, D., Klawonn, F., Kruse, R., Klawonn, F.: Foundations of Neuro-Fuzzy Systems. John Wiley & Sons, New York (1997)
4. Duch, W., Grudziński, K.: Prototype based rules - new way to understand the data. In: IEEE International Joint Conference on Neural Networks, Washington D.C, IEEE Press (2001) 1858–1863
5. Duch, W., Blachnik, M.: Fuzzy rule-based systems derived from similarity to prototypes. In Pal, N., Kasabov, N., Mudi, R., Pal, S., Parui, S., eds.: LNCS. Volume 3316. Physica Verlag, Springer, New York (2004) 912–917
6. Kuncheva, L.: On the equivalence between fuzzy and statistical classifiers. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems **15** (1996) 245–253
7. Wilson, D.R., Martinez, T.R.: Value difference metrics for continuously valued attributes. In: Proceedings of the International Conference on Artificial Intelligence, Expert Systems and Neural Networks. (1996) 11–14
8. Grąbczewski, K., Duch, W.: Heterogenous forests of decision trees. Springer LNCS **2415** (2002) 504–509
9. Blachnik, M., Duch, W.: Prototype-based threshold rules. LNCS **4234** (2006)
10. Merz, C., Murphy, P.: UCI repository of machine learning databases (1998-2004) http://www.ics.uci.edu/∼mlearn/MLRepository.html.
11. Quinlan, J.: C 4.5: Programs for machine learning. Morgan Kaufmann, San Mateo, CA (1993)
12. William, C.: Fast effective rule induction. In: Twelfth International Conference on Machine Learning. (1995) 115–123
13. Rapid-I: Rapidminer. http://www.rapid-i.com (-)
14. Weston, J., Elisseeff, A., BakIr, G., Sinz, F.: The spider. http://www.kyb.tuebingen.mpg.de/bs/people/spider/ (-)