Intelligent Data Analysis 5 (2001) 227–244
IOS Press

Hybrid systems of local basis functions

Ricardo Bezerra de Andrade e Silva[a] and Teresa Bernarda Ludermir[b]
[a]*Center for Automated Learning and Discovery, Carnegie Mellon University, 5000 Forbes Avenue, Pittsburgh, PA, USA*
E-mail: rbas@cs.cmu.edu
[b]*Centro de Informática, Universidade Federal de Pernambuco, Av. Professor Luis Freire s/n, Cidade Universitária, 50740-540 Recife, PE, Brazil*
E-mail: tbl@cin.ufpe.br

Received 12 September 2000
Revised 5 November 2000
Accepted 5 December 2000

Abstract. Since there is no individual approach that can be universally applied to effectively solve the hard problems of artificial intelligence and data analysis, hybrid systems are necessary to better tackle specific tasks by exploiting the advantages of different methodologies in a single framework. Based on known results of combining neural networks and rule-based systems, this work presents a hybrid system with the purpose of simplifying rule sets obtained from rule induction algorithms on classification problems without increasing the accuracy error. This is motivated by assuming that simplicity can lead to more understandable models and rule induction algorithms often provide an excessive number of rules necessary to classify future examples within a given accuracy error, even after pruning. Experimental evidence suggests effective gains on a benchmark of sixteen data sets. Experiments were also performed to detect the effect of different components of the proposed approach in achieving the results and so helping to explain why this hybrid system works.

Keywords: Hybrid systems, Occam's razor, classification

1. Introduction

Intelligent hybrid systems combine different methods for the solution of problems of artificial intelligence. It is difficult to determine accurately the degree in which parts of a problem solver could be considered "different approaches". As a result, the concept of hybrid systems is itself ill-defined. Those systems must integrate many "intelligent", "sophisticated" techniques, but even those concepts change in time [14]: well-succeeded intelligent techniques gradually become standard computational procedures.

It is easy to imagine systems in which the boundaries of categorization between hybrid and non-hybrid solutions are very imprecise: one can consider a case-based reasoning system that uses a loose heuristic to find an initial set of suitable cases and a more precise but computationally expensive procedure to refine this set and return the best of the selected solutions. Would this strategy be considered a hybrid one? As a result of the imprecise definition of hybrid systems, there are many different ways of categorization, varying from aspects of implementation to more abstract criteria (for example [23]). One can assume there is a whole continuum of categories, but as a general rule it is desirable that hybrid systems have a considerable level of complexity and integration.

Some concerns must be observed: hybrid systems are getting an increasing amount of attention from researchers in artificial intelligence and related fields, but there are reasons to generate skepticism [15]: many works seem to combine methodologies in an ad-hoc fashion, with no concerns in verifying the real advantages of those hybridizations. A hybrid system can achieve a good performance, but it does not mean that the basic techniques, when applied in isolation, are not better.

With those concerns on mind, one is motivated to create hybrid systems for machine learning and data mining by various reasons. This work focuses on transforming available classifiers into more comprehensible ones by removing several of its parameters on a data-driven approach, without necessarily losing predictive accuracy. In Section 2, the general idea and motivation is given. Section 2.1 describes how a given rule-based classifier is transformed into another representation and how one can interpret it. Section 2.2 describes the processes used to simplify the model while trying to keep the same accuracy or even increasing it. Section 3 discusses some experimental results, while Section 4 provides a discussion of related work.

2. A hybrid system for rule simplification

According to Domingos [7], the real role of Occam's razor as a criterion for model selection is choosing the most comprehensible model from those that achieve the same generalization error. Comprehensibility is desirable for providing insight about the process that produced the observed data, where the most noticeable application is aiding knowledge discovery in databases and analytical validation of automatically generated expert systems. For example, it is common that data mining processes generate large rule sets, resulting in incomprehensible models. In this case, the unintelligibility does not come from the knowledge representation language, but from the complexity of the rule set seen as a whole.

It is not easy to define the ideal properties of comprehensible models. This concept may vary under different domains and performance measures. For example, the structure of a model can be very complex, but very comprehensible because it is only slightly different from currently widespread knowledge.

However, a common measure of understandability is the structural complexity of the model. This can be determined as the number of parameters of the system, such as the number of premises of a rule base or the number of weights of a neural network. While not ideal, this criterion has the merit of being easily and generally applicable. It can be useful, for example, if it is used to discover that some pieces of knowledge are unnecessary to explain a phenomenon by simply dropping it and verifying that it is still possible to model the available data with high accuracy.

Under the context of hybrid systems, this paper proposes modifications in a classical neural network and statistical model for processing sets of classification rules automatically generated by rule induction algorithms. The motivation is to produce, by optimization techniques common in neural network learning, a reasonably reduced number of rules and premises. The aim is on interpretation, not on prediction, but of course one can achieve highly simplified models if a decrease on predictive performance is allowed. So, we also state that it is highly desirable that classification accuracy does not decrease in this process.

Traditionally, pruning techniques are necessary and widely used, but they may not be sufficient. For a more effective model simplification, steps of rule smoothing and fitting were added. The proposed approach consists in remapping a set of Boolean rules in a (neural) network of smooth kernels and then performing consecutive cycles of pruning and fitting until no further simplification can be performed using the same data sets applied in the rule induction stage. Table 1 illustrates this process. The final model should have classification accuracy comparable to the original rule base. A comparable accuracy must be understood as an accuracy rate that is not different by a statistically significant margin

Table 1
A proposed framework for rule set simplification

Task: find a simpler model with comparable classification accuracy
1. Parameterize a given rule base as a kernel-based neural network;
2. Following a determined order criterion, tentatively eliminate each unit from the network. If the error in the pruning set increases, put the unit back in the network. Similarly, perform a pruning of premises;
3. Optimize the network with a gradient-based learning algorithm (gradient descent, conjugate gradients, etc.);
4. Perform pruning as described on Step 2;
5. If in Step 4 we pruned some component of the model, go back to step 3. Otherwise, stop the process and output the network.

(according to a given hypothesis test) or by a practical margin (according to domain knowledge). The other assumptions underlying the applicability of this model are:

– the classification task that must be performed would significantly benefit from the generation of more comprehensible models;
– simplicity can be used as a measure of comprehensibility;
– the language used to reorganize the given rules is at least as comprehensible as the original propositional representation and may in fact give more insight about the model.

The meaning of this last statement must be clearly stated: the modified model is not a logical theory anymore, since its components are not independent Boolean rules. Instead, they must be interpreted as a mixture of probability density functions. Section 2.1.2 will discuss this issue in deeper detail.

An appropriate rule induction algorithm for initialization of the proposed model should identify rules by boxing subregions of the input space whose concentration of correctly labeled data is high. Rules induced by similar algorithms tend to correspond to the "bumps" of a multimodal distribution and so are more similar to kernel-based approaches for classification. Following Friedman and Fisher [11], we call our framework Bump Pruning and Fitting (BPF).

2.1. Mapping from Boolean rules to smooth kernels

The proposed model refines a set of classification rules specified in propositional logic using attribute-value records of a training data set. Numerical and multi-valued symbolic attributes are allowed on premises of each rule. The model can be graphically represented as a feed-forward neural network with one hidden layer of local basis functions, each hidden unit corresponding to one rule. The output layer uses one unit for each class of the domain and each hidden unit is linked to only one of the classes, in a similar way to a probabilistic neural network [32].

The following sections describe how the parameterization of the network is done and how it is initialized by a set of rules. More detailed information can be found in Silva [29].

2.1.1. Parameterization of individual units

Each hidden unit is built from a specific rule r as a local base function. This local function is parameterized by:

– a measure of center $\mu_r = (\mu_{r1}, \mu_{r2}, \ldots, \mu_{rd})^T$, where d is the dimensionality of the input space. This parameter is fixed during the learning stage;
– two measures of dispersion, $\mu_{r+} = (\mu_{r1+}, \mu_{r2+}, \ldots \mu_{rd+})^T$ and $\sigma_{r-} = (\sigma_{r1-}, \sigma_{r2-}, \ldots \sigma_{rd-})^T$. These are the free parameters adjusted during the learning stage.

Even using local measures of variability, the number of free parameters remains reasonably limited in the proposed model, because the vectors σ_{r+} and σ_{r-} are highly sparse: just a small subset of the whole set of domain variables is normally considered in each unit, as it is explained in Section 2.1.2. For those variables considered irrelevant in unit r, the dispersion parameter is represented as ∞. Note that this is one of the reasons for choosing two dispersion parameters σ_{r+} and σ_{r-}: many of the variables are bounded only by an upper or lower limit.

The activation function $a_r(x)$ for a hidden unit r given an input x is:

$$a_r(x) = \prod_k a_{rk}(x_k) \tag{1}$$

where $a_{rk}(x_k)$ is a function of attribute k at unit r.

The definition of the univariate function $a_{rk}(x_k)$ depends on the type of the variable x_k. For ordered (numerical) variables x_k:

$$\begin{aligned} a_{rk}(x_k) &= e^{-\frac{(x_k-\mu_{rk})^2}{\sigma^2_{rk+}}}, \text{ if } x_k > \mu_{rk} \text{ and } \sigma_{rk+} \neq \infty \\ &= e^{-\frac{(x_k-\mu_{rk})^2}{\sigma^2_{rk-}}}, \text{ if } x_k < \mu_{rk} \text{ and } \sigma_{rk-} \neq \infty \\ &= 1, \text{ otherwise} \end{aligned} \tag{2}$$

This is also similar to many traditional models of fuzzy neural networks [19].

For symbolic variables k, the measure μ_{rk} is itself a vector,[1] because multi-valued tests for categorical attributes are allowed in the proposed model. So, $a_{rk}(x_k)$, where $x_k = v$, is defined for unordered (symbolic) variables as

$$\begin{aligned} a_{rk}(x_k) &= e^{-\frac{(1-\mu_{rk,v})^2}{\sigma^2_{rk+,v}}}, \text{ if } \mu_{rk,v} < 1 \text{ and } \sigma_{rk+,v} \neq \infty \\ &= 1, \text{ otherwise} \end{aligned} \tag{3}$$

Each unit in the hidden layer is connected to exactly one unit in the output layer: the one that corresponds to the class of the original rule from where the hidden unit was built. The parameters of this final layer are not modified by gradient learning: the weights w_r, $r \in \{1, \ldots, R\}$, where R is the number of hidden units, are computed according to the behavior of unit r given the training data. This measure must penalize the less reliable units. There are similar concerns on developing rule induction algorithms. For selecting rules during the learning process and solving conflicts of overlapping rules in inference, one of the simplest measures is the apparent accuracy:

$$\text{Apparent accuracy } (r) = \frac{n_+}{n_+ + n_-} \tag{4}$$

where n_+ is the number of training examples that are of the same class of the rule and are covered by it. Equivalently, n_- is the number of training examples that are of a different class and are covered by the rule.

[1] $\mu_{rk,v}$ represents the vth component of this vector.

In fact, for selecting rules during induction, this measure is not appropriate because a rule covering a single item has perfect accuracy, as discussed by Clark and Boswell [5]. The rule induction algorithms used in the experiments of Section 3 apply a measure that trades-off coverage and accuracy. For inference purposes, the apparent accuracy measure is quite robust, especially if you consider computing the classification by combining the contribution of each rule and pruning small disjuncts that do not contribute for better prediction.

For the smooth rules used in the proposed model, a suitable measure is:

$$\text{Weighted accuracy}\ (r) = \frac{\sum_i I(C(x_i)) = C(r))a_r(x_i)}{\sum_i a_r(x_i)} \tag{5}$$

where $I(C(x_i) = C(r))$ is the indicator function for comparing the class of an example x_i and a unit r. It returns zero if x_i is not from the same class of r. The sum varies through all examples i.

The activation function of unit y_j of the output layer, where $C(y_j) = j$, is computed by:

$$y_j(x) = \frac{\sum_r I(C(r) = j)w_r a_r(x)}{\sum_r w_r a_r(x)} \tag{6}$$

where w_r is the link between r and j as measured by Eq. (5).

2.1.2. Initialization

Let P_r be the set of premises of rule r. P_{rk} is a premise concerning the k-th attribute in rule r. For ordered (numerical) variables, P_{rk} is represented by one of the following templates: $(a < x_{rk} < b)$, $(x_{rk} < b)$ or $(x_{rk} > a)$. For symbolic variables, P_{rk} assumes the template ($x_{rk} \in \text{Values}_{rk}$), where Values_{rk} is a subset of the domain of x_k. If x_k is not constrained in rule r (i.e., there is no premise concerning x_{rk}), then $P_{rk} = \varnothing$.

Unordered (symbolic) variables are represented in a 1-of-n encoding [26]: $x_{rk,v}$ corresponds to the function $I(x_{rk} = v)$, where $I(x)$ is the indicator function, returning 1 if x is true and 0 otherwise; $v \in \{1, 2, \ldots, n\}$ represents the v-th possible value of x_k, and n is the dimensionality of the domain of x_k.

Let D_r be the set of examples in the training set bounded by the corresponding rule r, where it is considered only those examples that are of the same class of r. For ordered variables, the parameters of the hidden units are initialized as follows:

- the center parameters μ_{rk}, computed for each attribute k, are given by the median of x_{rk} in D_r;
- two spread parameters, σ_{rk+} and σ_{rk-}, are computed for each attribute k. The initialization is $\sigma_{rk+} = \sqrt{2}|b - \mu_{rk}|/s$ and $\sigma_{rk-} = \sqrt{2}|\mu_{rk} - a|/s$. Reasonable values of s are usually 2 or 3, as explained later in this section. If $P_{rk} = \varnothing$, we consider $\sigma_{rk+} = \sigma_{rk-} = \infty$. If $P_{rk} = (x_{rk} < b)$ or $P_{rk} = (x_{rk} > a)$, $\sigma_{rk-} = \infty$ or $\sigma_{rk+} = \infty$, respectively;

For unordered variables, the parameters of the hidden units are initialized as follows:

- for all values v in the domain of x_k, if $v \in \text{Values}_{rk}$, then $\mu_{rk,v} = 1$. Otherwise, $\mu_{rk,v} = 0$;
- for the initialization of the spread parameters, there are two situations:
  - if $\mu_{rk,v} = 0,\ \sigma_{rk+,v} = \sqrt{2}/s,\ \sigma_{rk-,v} = \infty$. Typical values for s are 2 or 3;
  - if $\mu_{rk,v} = 1,\ \sigma_{rk+,v} = \sigma_{rk-,v} = \infty$;

Finally, each hidden unit must be introduced in the network by linking it to an appropriate output unit. The class C_r of the rule defines this relationship: the unit is linked only with the output unit representing C_r.

The initialization of the spread parameters is motivated by the normal probability distribution: nearly all of data fall in a distance of a few standard deviations of the center. The hard boundaries of a rule are relaxed by considering that most, not all, of the data falls in the hyper-box defined by the premises. The space between the center and a boundary is considered as a distance of two or three standard deviations, which corresponds to s in the definitions above. Slightly more overlapping is obtained by setting $s = 2$. The $\sqrt{2}$ factor is a small correction[2] introduced because the normal density function $\exp(-(x-\mu^2)/2\sigma^2)$ uses the dispersion parameter multiplied by 2. Note that the activation function is not a normal density function: asymmetric dispersions are considered and the median is used as a measure of center. Using the median instead of the mean makes the parameterization more robust to outliers. The dispersion parameters are indeed affected by outliers, but the fitting procedure (Section 2.2) is expected to correct any undesired influence by supervised learning.

As in the mixture of experts model [18], the BPF architecture can be interpreted as a combination of partial conditional probabilities $P(C|i, x)$, where i is the corresponding unit initialized from a rule, C is the class corresponding to unit i and $P(C|i, x)$ is simply the activation function $a_i(x)$ under the assumption of local variable independence. However, the mixing coefficients Eq. (5) are not a function of x itself, as it would be necessary to model $P(C|x)$. This simplification may lead to less accurate predictions, but requires much less free parameters and facilitates creating an order criterion for pruning purposes, as described in Section 2.2.

Concerning comparison with the original rules given as input, this approach clearly produces models that are not logical theories: any decision computed by BPF networks must be interpreted as a combination of uncertain pieces of evidence. There is no "rule extraction" step that translates this network of kernels back to Boolean representations. Frequently, rule extraction techniques must generate increasingly more complex models to get closer to the prediction accuracy of the original representation, and it is up to the modeler decide the appropriate trade-off.

Allowing overlapping of rules may sound undesirable if our goal is improved understandability of a model [16]. However, this is a common feature of pruning algorithms [12] and in fact is one of the reasons why it is possible to obtain a smaller set of rules with the same predictive power. Mechanisms such as decision lists are just conventions for handling rule conflict, not a solution to avoid the fact that rules will usually overlap. Futhermore, most kernels do not contribute significantly for the classification of a given point, and so one may consider only a small sets of probabilistic rules when trying to explain the output obtained. According to the experiments described in Section 3.3, there is evidence this is a likely outcome.

2.2. Pruning and fitting

A straightforward approach to pruning of parameters is used (as suggested for example by Tresp et al. [34]):

– prune basis functions: all hidden units (probabilistic rules) are organized in a crescent order of output weight. The pruning procedure tentatively removes the unit from the network and evaluates

[2] In the experiments described in Section 3, it gave slightly lower initial training errors. It must be clear that this is only a theoretical observation: this constant can be absorbed in the value defined for s.

the classification error. If the error on a pruning data set does not increase (it may decrease), the unit is permanently removed. This process is repeated for each unit according to the given order;
– prune premises: a similar procedure is applied, when all premises are ordered according to a simple sensitivity analysis criterion. Removing a premise is performed by setting its respective spread parameter to infinite;
– repeat this process till no basis function nor any premise can be pruned;

The order criterion for premises is a linear function that computes a heuristic measure contribution of each dispersion parameter. The contribution is defined by the change in the weight of a rule when a given spread parameter is temporarily discarded (i.e., set to infinite). Let w be the weight of a given unit and w_σ the weight of the same unit when parameter σ is discarded. Then, we have

$$\text{EvaluateSigma}\ (\sigma) = (w_\sigma - w)/w \tag{7}$$

which returns higher values for those dispersion parameters that when discarded achieved greater improvements (or smaller losses) for the weight of its respective unit.

As discussed by Breslow and Aha [3], more elaborated pruning techniques of rule bases and trees often require the transformation of the rules in a different representation, like graphs or rules with exceptions. This would make the comparison of BPF and the original model hard to perform. For example, one cannot directly compare the complexity of a tree and a rule set, because even though a tree can be transformed in a rule set, it would not be a fair comparison: a single node on a tree defines a premise for two rules, so it would be counted twice, when under the original tree representation it is represented by a single node. On the other hand, the hierarchical structure of tree may make it more confusing to understand when compared with relatively independent rules. Unless you want to face psychological considerations of different representations, which we avoid by explicitly assuming that comprehensibility will be measured by simplicity, it is desirable to follow conventional pruning techniques.

The straight approach adopted in this paper has similar counterparts like reduced error pruning (REP [3]). It must be clear that order of evaluation for units and spread parameters is just a heuristic measure for guiding the search in a model space (to avoid, for example, the computational complexity of REP). It does not intend to produce the smallest possible model with respect to a given data set. It is also useful when there is a huge quantity of parameters and one wants to cut the search at a given point, assuming that units with the smallest weights and most sensitive spread parameters are more likely to be eliminated.

The fitting procedure consists in computing the gradient of an error function with respect to the training set and using this information to update the vector of parameters. The error function utilized was the mean squared error (MSE). Although Cherkassky and Mulier [4] point that the MSE is not ideal for classification problems since a lower MSE does not guarantee a lower classification error, this is not a major concern with BPF, because the generalization error remains approximately the same during the fitting process. The weights in the output layer are adjusted after each update of the parameter vector. This is a variation of an expectation-maximization algorithm.

3. Experiments

Lim et al. [21] performed one of the most recent benchmarks comparing classifiers, focusing on decision trees, neural network-inspired algorithms and statistical methodologies. Their purpose was to extend the available results by adding more recent algorithms and emphasizing other three important

Table 2
This table summarizes information concerning the data sets studied in Lim et al. The first column ("Id") is the identifier that will be used along this section to refer to a given data set. The second field ("Name") is a short description of the modeled problem. For more details, consult the reference. "#Attrib" gives the total dimensionality (i.e., the number of attributes) of the respective problem, while "#Numeric" is the number of numeric attributes only. "#Cl" is the number of classes. The last two columns show the number of instances used for training and for testing. "10-fold cv" stands for 10-fold cross-validation procedures.

Id	Name	#Attrib	#Numeric	#Cl	#Train	#Test
bcw	Wisconsin breast cancer	9	9	2	683	10-fold cv
bld	BUPA live disorders	6	6	2	345	10-fold cv
bos	Boston housing	14	13	3	506	10-fold cv
cmc	Contraceptive method choice	9	2	3	1473	10-fold cv
hea	StatLog heart disease	13	7	2	270	10-fold cv
led	LED display	7	0	10	2000	4000
pid	PIMA indian diabetes	7	7	2	532	10-fold cv
sat	StatLog satellite image	36	36	6	4435	2000
seg	Image segmentation	19	19	7	2310	10-fold cv
smo	Attitude towards smoking	8	3	3	1855	1000
tae	Teacher assistant evaluation	5	1	3	151	10-fold cv
thy	Thyroid disease	21	6	3	3772	3428
veh	StatLog vehicle silhouette	18	4	4	846	10-fold cv
vot	Congressional voting records	16	0	2	435	10-fold cv
wav	Waveform	21	21	3	600	3000

aspects, one of them the complexity of the induced trees, where complexity was defined by the number of nodes obtained in the experiments.

Most of the data sets of this benchmark are from the UCI Machine Learning Repository [2].[3] Sixteen data sets were used, plus a replication of all of them with addition of attribute noise. A short description of the most important features of fifteen of those data sets is presented in Table 2.[4]

Some observations concerning these data sets: all records containing missing values were eliminated. The processed `pid` data set has one less attribute than the original specification that usually appears in the literature (for example [28]). Problems that originally used non-uniform misclassification costs (`hea`, for instance) are treated with uniform ones. There are two artificial data sets: `led` and `wav`. Most data sets were evaluated using 10-fold cross-validation as indicated. For more details and references about the data sets, see [21].

3.1. Experiments using PRIM, C4.5RULES and BPF

In this section, we discuss the results obtained using rules induced by PRIM [11] and C4.5RULES [27] on the same data sets presented in the previous section. PRIM is suitable as an initialization algorithm for BPF because it searches for salient subspaces where the average of a given function (Equation (4) in our experiments) is higher than in the whole input space. This results in a rule set that is more similar to a mixture of unimodal distributions.

We compare these algorithms with BPF initialized by PRIM (BPF-P) and BPF initialized by C4.5RULES (BPF-C). It is important to stress that PRIM and C4.5RULES rule sets were also post-

[3] At the current time of this writing, all data sets, with the exact split in folders for cross-validation or hold-out samples used in the experiments, are available at http://www.recursive-partitioning.com.

[4] The data set `dna` was excluded because the available implementation of PRIM, used in our experiments, does not handle domains with more than forty attributes. Also, we did not use the replicated data sets with injection of noise.

Table 3
Comparison of propositional rule sets and probabilistic rule sets refined by a BPF framework. The complexity measure is the number of premises of the model. For data sets where cross-validation was applied, the results were averaged over the 10 trials. Numbers in italic represent the standard variation of these results. Accuracy results were given to provide evidence that this approach is able to effectively reduce the complexity of the rule sets while maintaining similar classification power in nearly all data sets

Set	Classification error								Model complexity							
	PRIM		C45R		BPF-P		BPF-C		PRIM		C45R		BPF-P		BPF-C	
bcw	0.04	0.03	0.04	0.03	0.04	0.03	0.04	0.02	7.9	1.9	7.6	3.0	5.5	1.5	5.3	1.8
bld	0.34	0.08	0.31	0.05	0.31	0.06	0.35	0.05	24.9	4.4	23.4	7.7	15.9	5.9	10.6	4.9
bos	0.24	0.04	0.24	0.05	0.25	0.03	0.25	0.05	35	7.3	26.2	7.8	18.7	4.8	14.1	3.9
cmc	0.45	0.03	0.45	0.03	0.46	0.04	0.45	0.03	143	16.7	114	31.5	42.7	16.2	42.6	18.7
hea	0.15	0.07	0.20	0.07	0.16	0.05	0.19	0.08	16.4	4.9	17.9	6.6	8.2	3.2	11.2	3.1
led	0.27	–	0.28	–	0.28	–	0.27	–	58	–	50	–	29	–	35	–
pid	0.23	0.04	0.22	0.06	0.24	0.04	0.22	0.04	24.1	8.9	15.6	6.9	10.7	4.3	8.2	5.2
sat	0.14	–	0.14	–	0.13	–	0.13	–	292	–	212	–	165	–	125	–
seg	0.05	0.01	0.04	0.01	0.05	0.01	0.07	0.01	78.8	11.6	68.7	9.3	41.3	8.8	40.0	9.9
smo	0.31	–	0.31	–	0.31	–	0.31	–	10	–	14	–	12	–	16	–
tae	0.52	0.12	0.51	0.12	0.55	0.08	0.48	0.14	7.1	3.2	94.3	11.0	6.4	3.7	50.9	19,0
thy	0.01	–	0.01	–	0.02	–	0.03	–	16	–	8	–	9	–	3	–
veh	0.31	0.04	0.27	0.05	0.29	0.05	0.28	0.03	91	12.2	55.9	13.6	51.2	7.8	34.6	11.6
vot	0.06	0.04	0.05	0.03	0.05	0.03	0.04	0.03	5.1	2.5	12.1	3.2	4.2	2.1	7.10	3.4
wav	0.26	–	0.25	–	0.21	–	0.24	–	67	–	77	–	40	–	41	–

pruned by the same pruning policy, described in Section 2.2 using the training set as the pruning set: for instance, the original rule sets produced by C4.5RULES were much larger than those reported on this section. All accuracy results on Table 3 were measured with respect to pruned rule sets. The BPF variations were initialized by a remapping of the final models obtained using PRIM and C4.5RULES after pruning. This was performed to verify if the apparent extra simplification obtained by BPF is a result not only of the pruning.

The optimization procedure used for adjustment of parameters was the conjugate gradients algorithm. The data was normalized to a 0–1 range to increase stability of the optimization method. In all experiments, the stop criterion for the fitting stages was convergence of the conjugate gradients procedure.

The metric of model complexity was the number of premises in a rule set. The results obtained show that BPF variations were able to deliver more simplified rule sets in all studied data sets, but this does not mean that it will happen in all situations. BPF-C seems to be a winner compared with BPF-P, but one must consider that BPF-C was a bit less stable: for the data sets `bld`, `seg` and `thy`, it indeed increased the classification error rate by a margin bigger than 1% (or, in the case of `thy`, three times larger), while BPF-P had problems with `tae`. The average ratio of simplification[5] for BPF-P was 59.9% with a standard deviation of 0.097, while BPF-C achieved a ratio of 55.05% with a standard deviation of 0.089. Some special care must be taken regarding comparison of BPF-P and BPF-C: in BPF and PRIM, premises involving symbolic attributes add a parameter for each value that is not present in the original rule premise. This means that in data sets like `tae`, where there are symbolic attributes with nearly thirty values, a typical premise of C4.5RULES is counted as having almost thirty parameters. However, it must be stressed that in this benchmark, only `hea`, `led`, `smo`, `tae` and `vot` have symbolic variables, where `led` and `vot` have only binary variables (which makes no difference when comparing PRIM with

[5]This ratio is defined as the number of parameters of BPF divided by the number of parameters of the corresponding PRIM and C4.5RULES pruned rule sets. The data sets `cmc` and `smo` were treated as outliers and excluded from the computation of this statistic, because no algorithm was able to achieve a classification error less than the default rule.

Table 4
The differences in classification error on train and test sets for pruned PRIM rule sets and a BPF variation not yet optimized. Note that the highest differences are on the training sets, not the test ones. The differences in training sets for a not pruned PRIM rule set and a BPF representation are usually lower

	PRIM		BPF-P			PRIM		BPF-P	
	train	test	train	test		train	test	train	test
bcw	0.02	0.04	0.05	0.05	seg	0.02	0.05	0.09	0.09
bld	0.18	0.30	0.26	0.31	smo	0.29	0.334	0.30	0.312
bos	0.14	0.24	0.23	0.29	tae	0.44	0.55	0.52	0.53
cmc	0.34	0.46	0.41	0.45	thy	0.004	0.01	0.024	0.03
hea	0.11	0.16	0.14	0.19	veh	0.13	0.30	0.27	0.33
led	0.24	0.27	0.31	0.33	vot	0.03	0.05	0.03	0.05
pid	0.14	0.24	0.20	0.25	wav	0.09	0.22	0.18	0.21
sat	0.06	0.13	0.11	0.14					

C4.5RULES) and smo is not a useful data set to compare, because the classifiers did not perform better than the default rule.

Notice that in data set smo, plain PRIM produced less parameters than BPF-P itself. This is possible because the PRIM and C4.5RULES variations used in this experiments classify a new input according to the class of the rule that first matches it, following a decision list organized in a decreasing order of rule weight. The pruning performed under this approach usually results in smaller rule sets than under the policy that combines rule results in a neural-network style. This last variation was adopted to post-prune rule sets that were used to initialize BPF-P and BPF-C (and so they may have more parameters), because in principle this initial pruning is performed in a rule set whose functional form is more biased towards the neural network representation. However, in practice, this additional care may be unnecessary.

3.1.1. Explaining gains

This subsection discusses some reasons why BPF works consistently well for simplification of rule sets and how it could behave with different pruning policies. It also evaluates its sensitivity to the initialization procedure.

Rule induction algorithms usually generate overlapping regions in the input space, which result in redundant components that can be successfully eliminated from the model with no harm to its predictive capabilities. But there are many situations when even a pruned Boolean rule base can be simplified when reparameterized as a local basis function network: examples that are uncovered when a given rule is eliminated are correctly classified by others because the smoother activation functions Eq. (1) are able to cover them.

This justifies the cycle of pruning and fitting indicated on Table 1, where retraining is done after any model simplification: some units are fitted in a way to subsume the roles of other units. So, it is expected that there is the possibility of further pruning after a session of model refinement. Pruning makes those rules cover more of the input space. This increases the chance of a bias mismatch between the original rule set and the network of smooth kernels, having a negative impact on the described initialization procedure. However, the cycle of pruning and fitting performed by BPF makes this model more robust to violations of the assumption of similarity. Table 4 shows a comparison of PRIM results and an initialized, but not optimized, BPF. Note that the biggest differences are on the training sets, not the test ones. In fact, the initial difference in test set errors and training set errors is much lower in untrained BPF models than in the original rule-based classifier. This provides evidence for part of the answer to why BPF successfully simplifies rules: it ignores some of the overfitting of the rules by smoothing out irrelevant

details. The degree of smoothness is controlled by the parameter s, as described in Section 2.1.2. This has a relation with adjustment of Parzen windows and the bias-variance trade-off [13].

After the pruning and fitting cycles, some of the training errors reached approximately the same of the original PRIM rule sets, while others converged to higher training errors, but with no damage to generalization. In the experiments performed with the discussed benchmark, BPF achieved a training error of 0.07 in `sat` and 0.10 in `wav`, which are basically the same achieved by PRIM, while getting higher error in `bos` (0.19) and `pid` (0.19), for instance.

Another explanation for the simplification gains of BPF comes from the nature of the fully supervised learning procedure embedded in this framework. Since the minimization of the training error is performed with respect to the network as a whole (unlike the separate-and-conquer process of recursive covering), BPF is able to redistribute its basis functions in a similar way by which fully supervised learning in RBFs works. This leads to the formation of redundant components that are later removed by pruning cycles.

One could argue that the applied pruning procedure did not fully explore possibilities of simplification for the rule-based classifier, since it used the same data sets for induction and pruning. Extra experiments were performed using a separated pruning set [30], and BPF still maintained a fair difference compared to the original Boolean representations.

3.2. Learning curves and model complexity

Usually, all classifiers achieve an asymptotic level of generalization accuracy as a function of the amount of data used for training. Learning curves are a useful way to visualize this process. These graphs plot the number of instances against the (average) resulting accuracy of a process of training. It is interesting to observe the behavior of model complexity under those asymptotic conditions. It is desirable that a learning model does not increase in complexity when it is not possible, under its fitting capabilities, to improve the generalization accuracy when more data is given.

Many non-parametric models like tree and rule inducers are especially sensitive to this problem, as pointed out by Oates and Jensen [25]. They present evidence of an approximate linear relationship between model complexity and number of available instances, regardless if accuracy increases or not, in a series of experiments involving popular algorithms. This leads to needlessly complex and cumbersome rule sets. Empirically, cost-complexity pruning methodologies, like those introduced by CART decision trees, which explicitly accept a trade-off between accuracy and complexity, are much less susceptible to this problem (but may result in higher generalization errors). The framework proposed in this paper does not intend to sacrifice predictive power. It is interesting to see how it behaves under the condition of increasing data sets.

We performed an experiment with one of the data sets used by Oates and Jensen, the led-24 domain. Its objective is to classify the image of a digit from a led display in one of ten classes (from 0 to 9), using 0-1 valued attributes, where each attribute indicates if the corresponding light of the led is on or off. In principle, it is a very easy problem, but to test the robustness of a classifier, 10% noise was added to each attribute. Also, 17 random attributes are added: their values are chosen within a uniform distribution not related with the class of the instance. The Bayesian optimal error is about 26% and according to the documentation available at the UCI Machine Learning Repository [2], it is considered a difficult problem.[6]

[6] As one should expect from the effects of the curse of dimensionality, it is interesting to notice that plain nearest neighbors algorithms (that do not use adaptive measures of distance) fail by a large margin to achieve reasonable results: the UCI Repository reports an error rate of 59% for a standard nearest neighbor model.

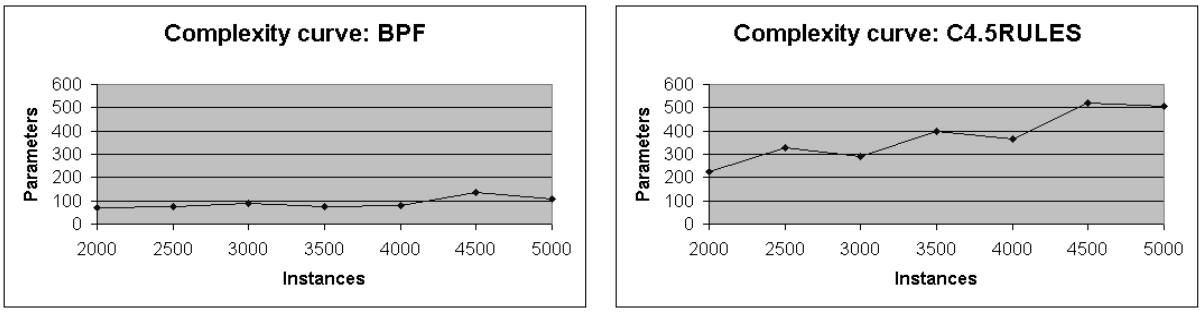

Fig. 1. Complexity curves for BPF and a post-processed C4.5RULES.

Table 5
Variation in the number of parameters as a function of the number of training instances. The column increase indicates the ratio between the correspondent number of premises with respect to the number of premises obtained with 2000 instances

	BPF		C4.5RULES	
	premises	Increase	premises	increase
2000 instances	70	–	226	–
2500 instances	76	8.6%	327	44.7%
3000 instances	90	28.6%	292	29.2%
3500 instances	73	4.3%	399	76.5%
4000 instances	78	11.4%	367	62.4%
4500 instances	137	95.7%	518	129.2%
5000 instances	109	55.7%	505	123.4%

Seven trials were performed with a varied amount of data for comparison of BPF and C4.5RULES. Figure 1 depicts a graphical comparison between the two classifiers. C4.5RULES sets were post-processed by the same pruning procedure, with an ordered decision list as the conflict policy. The mean error rate of BPF was 25.09%, with a standard deviation of 0.004. C45RULES achieved 27.62% and a standard deviation of 0.007. The low standard deviations are an indication that the error was nearly uniform in all trials, independently of the amount of data.

Even though BPF considerably simplifies rules obtained by C4.5RULES with a slight increase in classification accuracy, there is still some effect of linear increase in complexity. Table 5 details statistics obtained during the trials, with the corresponding increase in complexity when compared with the number of premises in the model induced with 2000 instances. There is a huge leap at 4500 instances, denoting that in principle there is still a bias favoring increase in complexity even under the BPF framework. However, the increase has different practical impact when one considers that jumping from 70 premises to 109 is less harmful for understandability purposes than jumping from 226 to 505 according to the absolute number of parameters.

3.3. More about rule quality: Weight distribution and small disjuncts

In this section, we evaluate how the BPF framework changes some characteristics of the rule sets given as input. The objective is to give insight about distribution of the rules, their locality characteristics and also other possible explanations of how BPF is able to carefully simplify a network of kernels without losing prediction accuracy.

Table 6
Weight statistics, showing mean and standard deviation of the average weight associated with the kernels of the network

	PRIM		BPF-P			PRIM		BPF-P	
	mean	sd	mean	sd		mean	sd	mean	sd
bcw	0.8635	0.0423	0.9369	0.0805	Seg	0.7143	0.0396	0.7795	0.1019
bld	0.6981	0.0307	0.6572	0.0467	Smo	0.4146	–	0.2478	–
bos	0.6613	0.0327	0.6940	0.0623	Tae	0.4764	0.0219	0.4030	0.0275
cmc	0.5466	0.0128	0.4772	0.0345	Thy	0.7539	–	0.4921	–
hea	0.7820	0.0315	0.7299	0.0543	Veh	0.6278	0.0195	0.6650	0.0740
led	0.4939	–	0.3951	–	Vot	0.8397	0.0347	0.7984	0.0550
pid	0.7162	0.0373	0.6922	0.0376	Wav	0.7745	–	0.8510	–
sat	0.6888	–	0.8092	–					

Moody and Darken [24] report experiments using full supervised learning for radial basis function networks (RBFs). They obtained kernels with very broad receptive fields. Since BPF uses less rules to express a mapping, one could suppose that the final model is more "distributed", or less "local", than the original one, i.e. what was previously classified by a single rule (or a small subset of rules) is now the result of a combination of a significantly larger set of kernels. For interpretability purposes, this may seem somewhat undesirable, although much less problematic than distributed representations in dictionary methods [4], such as RBFs, where the same components are used for every class. However, we still want to evaluate how kernels are weighted, since one of the motivations of not treating the output weights as adaptive parameters was to bias the kernels towards local coverage. Table 6 summarizes information concerning weight distribution for the benchmark discussed in Section 3.1. For a given training set divided in cross-validation folders, we took the average of the hidden unit weights in each trial, and then computed the mean and standard deviation of those trials.

The way weights are computed in PRIM is slightly different from BPF but the purpose of this comparison is to show that there is not a definitively better approach between these methods, and so this difference is not much relevant. PRIM wins in nine of the fifteen data sets, but it is not clear if this result could be extended to a larger sample of data sets. In fact, excluding the ill-defined `smo` (where normally a classifier converges to the default rule), PRIM wins in 8 of 14 data sets, or only 57% of the experiments. One cannot say that there is a clear dominant representation even in this benchmark, especially when considering that in some data sets (as `pid` and `vot`) the difference is relatively small.

The large difference in `thy` can be explained by the fact that in this data set the error rate is extremely low in a reasonable large data set. Some original rules of very high weight, but small coverage when compared to the size of the data set, were excluded by the smoothing process of initialization, significantly impacting the average weight of the final network produced by BPF. For the `led` data set, the Boolean rules already composed a model of low average weight. BPF followed this bias by making the mapping even more distributed. Both `tae` (much higher error rates than those achieved by CART, according to Lim et al. [21]) and `cmc` (much higher number of rules, even after pruning with BPF) are exceptional domains where PRIM bias was not appropriate, and hence weakens the relevance of weight distribution difference for those data sets.

A different approach to evaluate the quality of the induced model can be undertaken by observing the sensitivity of the probabilistic rules to the problem of small disjuncts. The problem of small disjuncts was formalized by Holte et al. [17]: rules with small coverage are highly error-prone. Most of the errors committed by rule-based classifiers (including decision trees) are due to rules that cover a small portion of the input space. For example, Holte et al. present a set of experiments where rules with individual coverage less than ten cases were responsible for up to ninety-five percent of the classification errors.

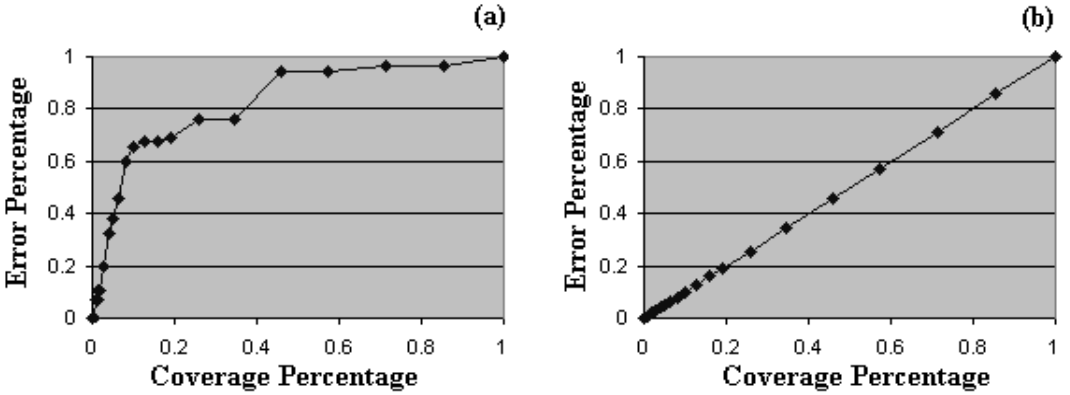

Fig. 2. In (a), we have a typical distribution of accumulative percentage of coverage against accumulative percentage of errors in a real-world data set (one of the training sets derived from data set seg of Section 3.1). Picture (b) shows an idealized distribution where the percentage of error is proportional to the accumulated coverage.

Collectively, those rules were able to match a reasonable amount of data, but it is still far from the proportion of committed errors.

Figure 2 depicts a typical graphic for analyzing this problem. One axis is proportional to the accumulated coverage, obtained by summing the coverage of individual rules in a crescent order of number of matched instances: i.e., for the partial coverage of the first j rules,

$$\text{Coverage}\,(r_i) = \text{number of instances covered by } r_i \tag{8}$$

$$\text{AccumulatedCoverage}\,(j) = \text{Coverage}\,\left(\bigcup_{i=1}^{j} r_i\right) \tag{9}$$

and if $k > j$, then Coverage $(r_k) \geqslant$ Coverage (r_j).

The other axis gives the percentage of all incorrectly classified instances that are covered by the j rules that correspond to the point of AccumulatedCoverage (j). For example, for a point where the accumulated coverage percentage (Eq. (9) divided by the total number of instances) is 0.45 and the error percentage is 0.70, it means that the smallest disjuncts that cover 45% of all training instances are responsible for 70% of all errors. In Fig. 2, 50% of coverage gives raise to more than 90% of all errors.

Note that small disjuncts are a major factor causing the increase in the number of components of a rule-based classifier. As stated in the beginning of this section, the next paragraphs will focus on the evaluation the quality of the probabilistic rules obtained: we are not trying to give any different treatment to small disjuncts in order to improve accuracy.

Following Ali and Pazzani [1], it is interesting to emphasize the role of weighted rules in measuring the small disjuncts effect. They concluded that allowing formation of overlapping disjuncts and choosing a class by a function of the weights of the rules that cover a given input lessen the impact of small disjuncts, as they demonstrate by a comparison using different relational learning models. For Boolean kernels as those obtained by PRIM, we define that a rule "covers" an instance if it has the highest weight among all rules that matches the given input. For BPF kernels, we define that a (probabilistic) rule covers an instance if the product of its activation and its weight is the highest among all rules. Ties are rare and in

Table 7
Summarized information concerning the distribution of accumulated coverage against accumulated errors for PRIM and BPF

	PRIM		BPF-P			PRIM		BPF-P	
	mean	sd	mean	sd		mean	sd	mean	sd
bcw	0.2247	0.0961	0.0898	0.1072	seg	0.3085	0.0421	0.2524	0.0595
bld	0.0498	0.0349	−0.0177	0.0327	smo	−0.0059	–	−0.0937	–
bos	0.1225	0.0207	0.0449	0.0369	tae	−0.0086	0.0188	−0.0197	0.0313
cmc	0.0346	0.0140	0.0169	0.0158	thy	0.3906	–	0.0923	–
hea	0.0832	0.0420	0.0467	0.0596	veh	0.1159	0.0336	0.1163	0.0338
led	0.0309	–	0.0449	–	vot	0.1356	0.0519	0.1770	0.1166
pid	0.0555	0.0339	0.0194	0.0362	wav	0.1233	–	0.0321	–
sat	0.1718	–	0.1215	–					

practice this criterion guarantees that instances are not counted twice. The accumulated error is just the number of instances that are covered by a unit that is from a different class.

To summarize the comparison of accumulated covering against accumulated errors, CE, we report the following statistic

$$CE(\text{model}) = \frac{1}{N}\sum_{j=1}^{N}\{\text{AccumulatedError}\,(j) - \text{AccumulatedCoverage}\,(j)\} \tag{10}$$

where N is the number of rules. AccumulatedError (j) is the percentage of instances incorrectly covered by the first j rules. AccumulatedCoverage (j) is defined in Eq. (9). Note that this statistic can be negative. It can be interpreted as an estimator proportional to the area above the diagonal of a Coverage $\times$ Error graphic (like Fig. 2) minus the area below the diagonal. Table 7 presents the results for the benchmark of Section 3.1 comparing PRIM and BPF. Note that this definition counts errors with respect to the units and, though similar, not necessarily corresponds to the accumulated errors of the whole mapping.

This measurement can be interpreted as another indicative of the quality of the rules. Since rules of small coverage commit fewer errors, and they are the majority, they are more likely to individually contribute for the classification without interfering with other kernels.

Also, distributions more similar to Fig. 2(a) than Fig. 2(b) tend to require more fragmented rule sets, implying more complex models (i.e., larger rule sets). However, this is only another partial factor that explains why BPF generates simpler models, since for some data sets (e.g., `veh`) there is no real difference in this statistic.

4. Related work

Networks of kernel functions are widespread in many applications of artificial intelligence, and its relation to rule-based knowledge is discussed at length in several publications (for example [19]). The greatest depart that the proposed approach takes is focusing on delivering sets of probabilistic rules that perform comparatively well in prediction accuracy while avoid unnecessary complexity.

Tresp et al. [34] presents one of the most similar frameworks. Among the differences, one has a distinct way to parameterize the individual kernels. Tresp et al. interpret kernels as unconditional distributions. Premises for numerical variables, for instance, are symbolically represented as $x_k = \textit{value}$, and are parameterized as normal distributions centered at value. BPF maps intervals such as $x_k < \textit{value}$ into conditional distributions.

Some of the first works about hybrid systems of neural networks and rules were surveyed in [33]. However, those frameworks focused mainly on multilayered perceptrons, and not local basis functions. The primary purpose was refining a given theory expressed in logical rules. One interesting advantage was allowing the output of a rule to be used as an input of another, instead of the straight input-output mapping that is common in kernel-based networks. Some variations included a step that translated the neural network back to a logical representation at a considerable computational cost.

Tree-based RBF [20] is very similar to BPF concerning the methodology of mapping from rules to smooth kernels. However, TB-RBF focuses on fitting the neural network in order to improve prediction accuracy. The free parameters are the output weights of the network, and each kernel is linked to all output units representing classes (even if the kernel is not related to the classes represented by such units). This full connection between rules and classes may not help to make the model more comprehensible, but Kubat's experiments gave evidence it was fundamental to improve its accuracy over the original rule base (in fact, a classification tree).

From another perspective, hybrids of networks of local basis functions and rule-based knowledge has been used for various purposes: Smyth et al. [31] use the partitions defined by classifications trees as a way to select features and diminishes the impact of the curse of dimensionality in Parzen windows estimators of probability. Similarly, Friedman et al. [10] applies a related combination for information retrieval. Wettschereck et al. [35] discusses many applications of gradient-based optimization for learning of local measures of attribute importance for instance-based models.

There is also a close relation with the RIPPER [6] methodology of rule optimization. RIPPER algorithm alternates steps of pruning, rule revision and rule addition to cover any remaining positive examples. This process is repeated k times, and those variants are called RIPPERk induction algorithms. There is no rule smoothing, which was one of the main factors that contributed for model simplification under BPF framework. Notice that one can use the final set of rules output by RIPPER as an input to BPF.

5. Conclusions

The BPF framework does not intend to offer new approaches for rule induction, but rather building on existing ones by allowing alternative techniques for model interpretation. In principle, any rule induction algorithm can be used as part of this hybrid system, even those that already perform repeated cycles of pruning.

Due to these various restrictions for interpretability warranties, in practice BPF hardly improves the classification accuracy for rules generated by adequate rule induction algorithms. There is no adaptive recombination of kernels, unlike the TB-RBF approach of Kubat [20]. There is no addition of extra components (i.e., rules and premises) to explicitly lower the classification error, which in fact would conflict with the purpose of model simplification. Indeed, many neurofuzzy approaches for classification (highly similar to BPF) can be criticized for failing to achieve reasonable accuracy improvements [15].

The algorithm is less useful for problems where noisy is a minor issue, or there are few variables or little data. Under these situations, rule induction algorithms will be less prone to generate unnecessary rules. Also, since the procedure relies on gradient-based optimization, it will require more computational time when compared with greedy rule induction algorithms, and one may want to apply some principled procedure to choose the smoothing factor (e.g., by cross-validation) to assure no loss of accuracy or even a trading-off predictive power for simplicity.

Some topics for future work are:

– scaling up. A straight implementation of gradient-based optimization is usually very slow for larger data sets, where "large" can be as little as 10,000 instances depending on the number of parameters. However, improvements can be achieved by taking advantage of the local characteristics of kernels. Omohundro [22] sketches some structures to rapidly evaluate basis functions that can be used as a starting point to the development of faster BPF algorithms;
– evaluating BPF as an approach for recovering from overpruning. Since some pruning algorithms may cause true losses in classification prediction [8], how the supervised learning would be able to recover the model from those losses without addition of new kernels, i.e., just by fitting the existing parameters? For example, by branching the process at a given point into separated networks (allowing some α% increase on the pruning set error), performing the fitting and returning the one with lowest classification error in a training/validation set. If networks with (values greater than zero are chosen, it will result in even simpler models. The computational cost of this beam search in a model space is higher, of course, but easily leads to parallel implementations;
– handling variance. The variance in the number of parameters after BPF remained high, inheriting it from the rule-based induction itself and the adopted pruning methodology. It would be interesting to experiment how different pruning methodologies [3] to reduce this variance;
– creating measures of rule surprisingness for knowledge discovery purposes [9]. One could investigate how the differences between the initial and final states of a given rule would be related to its degree of surprisingness. For example, rules of low coverage tend to be discarded as noisy or irrelevant components by the smoothing, pruning and fitting process. If they remain in the final rule base with little or no modification, they could be pointing to interesting subclasses of the domain.

Acknowledgements

This work was supported by a CNPq grant.

References

[1] K. Ali and M. Pazzani, Reducing the small disjuncts problem by learning probabilistic concept descriptions, *Computational Learning Theory and Natural Learning Systems* **3** (1992), 183–199, MIT Press.
[2] C. Blake, E. Keogh and C.J. Merz, *UCI Repository of machine learning databases*, University of California, Department of Information and Computer Science, Irvine, CA, URL: http://www.ics.uci.edu/ mlearn/MLRepository.html, 1998.
[3] L.A. Breslow and D.W. Aha, Simplifying decision trees: a survey, *Knowledge Engineering Review* **12** (1997), 1–40.
[4] V. Cherkassky and F. Mulier, *Learning From Data: Concepts, Theory and Methods*, Wiley Interscience, 1998.
[5] P. Clark and R. Boswell, Rule induction with CN2: some recent improvements, *Proceedings of the 5th European Conference on Machine Learning*, Springer Verlag, 1991, pp. 151–163.
[6] W.W. Cohen, Fast effective rule induction, *Proceedings of the 12th International Conference on Machine Learning* (*ML95*), Morgan Kaufmann, 1995, pp. 115–123.
[7] P. Domingos, The role of Occam's Razor in knowledge discovery, *Data Mining and Knowledge Discovery* **3** (1999), 1–19.
[8] F. Esposito, D. Malerba and G. Semeraro, A comparative analysis of methods for pruning decision trees, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **5** (1997), 476–491.
[9] A.A. Freitas, On objective measures of rule surprisingness, *Proceedings of the 2nd European Symposium on Principles of Data Mining and Knowledge Discovery*, Springer-Verlag, 1998, pp. 1–9.
[10] J.H. Friedman, J.L. Bentley and R.A. Finkel, An algorithm for finding best matches in logarithmic expected time, *ACM Transactions of Mathematical Software* **3** (1977), 209–226.
[11] J.H. Friedman and N.I. Fisher, Bump hunting in high-dimensional data, *Statistics and Computing* **9** (1999), 123–143.
[12] J. Fürnkranz, Separate-and-conquer rule learning, *Artificial Intelligence Review* **13** (1999), 3–54.

[13] S. Geman, E. Bienenstock and R. Doursat, Neural networks and the bias/variance dilemma, *Neural Computation* **4** (1992), 1–58.
[14] C.L. Giles, R. Sun and J.M. Zurada, Neural networks and hybrid intelligent models: foundations, theory, and applications, Guest editorial, *IEEE Transactions on Neural Networks* **9** (1998), 721–723.
[15] A, Gray, *Hybrid Systems FAQ*, URL: http://divcom.otago.ac.nz:800/COM/INFOSCI/SMRL/people/andrew/publications/faq/hybrid/parts/Info.htm, 1997.
[16] G.E. Hinton, J.L. McClelland and D.E. Rumelhart, Distributed representations, in: *Parallel Distributed Processing*, (Vol. 1), D. Rumelhart and J.L. McClelland, eds, MIT Press, 1986, pp. 77–109.
[17] R.C. Holte, L.E. Acker and B.W. Porter, Concept learning and the problem of small disjuncts, *Proceedings of the 11th International Joint Conference of Artificial Intelligence*, Morgan Kaufmann, 1989, pp. 813–818.
[18] R.A. Jacobs, M.I. Jordan, S.J. Nowlan and G.E. Hinton, Adaptive mixture of local experts, *Neural Computation* **3** (1991), 79–87.
[19] B. Kosko, *Fuzzy Engineering*, Prentice Hall, 1997.
[20] M. Kubat, Decision trees can initialize radial-basis function networks, *IEEE Transactions on Neural Networks* **9** (1998), 813–821.
[21] T. Lim, W. Loh and Y. Shih, A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms, *Machine Learning Journal* **40** (2000), 203–229.
[22] S.M. Omohundro, Bumptrees for efficient function, constraint and classification learning, Technical Report TR-91-009, International Computer Science Institute, Berkeley, CA, 1991.
[23] K. McGarry, S. Wermter and J. MacIntyre, Hybrid neural systems: from simple coupling to fully integrated neural networks, *Neural Computing Surveys* **2** (1999), 62–93. Eletronic publication, URL: http://www.icsi.berkeley.edu/~jagota/NCS/, 1999.
[24] J. Moody and C.J. Darken, Fast learning in networks of locally-tuned processing units, *Neural Computation* **1** (1989), 281–294.
[25] T. Oates and D. Jensen, Large datasets lead to overly complex models: an explanation and a solution, *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining*, AAAI Press, 1998, pp. 294–298.
[26] D. Pyle, *Data Preparation for Data Mining*, Morgan Kaufmann, 1999.
[27] J.R. Quinlan, *C4.5: Programs for Machine Learning*, Morgan Kaufmann, 1993.
[28] B.D. Ripley, *Pattern Recognition and Neural Networks*, Cambridge University Press, 1996.
[29] R.B.A. Silva, *Hybrid Systems of Local Basis Functions*, MSc. dissertation. Centro de Informática, Universidade Federal de Pernambuco. Recife-PE, Brazil, 1999.
[30] R.B.A. Silva and T.B. Ludermir, Obtaining simplified rule bases by hybrid learning, *Proceedings of the 17th International Conference on Machine Learning*, Morgan Kaufmann, 2000, pp. 879–886.
[31] P. Smyth, A. Gray and U.M. Fayyad, Retrofitting decision tree classifiers using kernel density estimation, *Proceedings of the Twelfth International Conference on Machine Learning*, Morgan Kaufmann, 1995, pp. 506–514.
[32] D. Specht, Probabilistic neural networks, *Neural Networks* **3** (1990), 109–118.
[33] G.G. Towell and J.W. Shavlik, Knowledge-based artificial neural networks, *Artificial Intelligence* **70** (1994), 119–165.
[34] V. Tresp, J. Hollatz and S. Ahmad, Representing probabilistic rules with networks of gaussian basis functions, *Machine Learning* **27** (1997), 173–200.
[35] D. Wettschereck, D.W. Aha and T. Mohri, A review and comparative evaluation of feature weighting methods for lazy learning algorithms, *Artificial Intelligence Review* **11** (1997), 273–314.