Obtaining Simplified Rule Bases by Hybrid Learning

Ricardo Bezerra de Andrade e Silva RBAS@DI.UFPE.BR
Teresa Bernarda Ludermir TBL@DI.UFPE.BR
Centro de Informática, Universidade Federal de Pernambuco, CP 7851, Recife PE - Brazil

Abstract

In many rule induction algorithms, it is relatively common that a large number of rules is generated. This complexity may harm the comprehensibility of the model without improving its predictive performance. For aiding automatic knowledge acquisition and knowledge discovery in classification domains, we propose a framework to post-process Boolean rules obtained from rule induction algorithms. This process generates probabilistic rule sets with fewer rules and premises, while maintaining comparable classification accuracy.

1. Introduction

According to Domingos (1999), the real role of Occam's razor as a criteria 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 observable data. This is specially useful for analytical validation of automatically generated representations (e.g., in knowledge acquisition for experts systems) and for knowledge discovery (Fayyad et al., 1996). Some trade-offs concerning decreases in performance by using more comprehensible models may be tackled.

However, it is not easy to define the ideal properties of comprehensible models. This concept may vary under different domains and performance measures. As also stated by Domingos, 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 criteria has the merit of being easily and generally applicable.

Following this principle, we propose a framework for simplification of propositional rules obtained from rule induction algorithms in classification problems. The 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 2.3 will discuss this issue in deeper detail.

Table 1. A framework for simplification of rule bases.

Task: generate a simpler model with comparable classification accuracy

1. Parameterize a given rule base as a network of smooth kernels;
2. Following a determined order criteria, tentatively eliminate each unit from the network. If the error in the pruning set increases by more than a small percentage, 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 a pruning analog to 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;

This 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 (according to a given hypothesis test) or by a practical margin (according to domain knowledge).

2. Model Representation and Initialization

Propositional rules can be interpreted as hyper-rectangular uniform kernels in a multidimensional space (Andrews & Geva, 1996). We propose mapping these uniform kernels in smooth kernels. The motivations underlying this decision are two-folded: it is a way to diminish the "brittleness" of this representation and making further simplification possible, as explored in Section 4; it allows the computation of the gradient of a given error function, which can then be plugged into gradient-based optimization algorithms. Unlike typical separate-and-conquer algorithms for rule induction (Fürnkranz, 1999), this optimization procedure computes the contribution of all parameters at the same time and it can be useful for redistribute the kernels according to an error function.

Before describing how the remapping is performed, the structure of the model is described in Section 2.1.

2.1 Representation

In a given classification rule r used to initialize this model, each premise i is a Boolean function that maps a pair (variable$_{ri}$, interval$_{ri}$) to true if and only if variable$_{ri}$ $\in$ interval$_{ri}$. If variable$_{ri}$ is a symbolic variable, interval$_{ri}$ is defined by a discrete set {value$_{ri1}$, value$_{ri2}$, ..., value$_{rim}$}. If variable$_{ri}$ is a numerical variable, interval$_{ri}$ is a continuous intervals [value$_{ri1}$, value$_{ri2}$]. Note that if one interprets the whole premise$_i$ as a propositional variable, a set used for initialization must be in disjunctive normal form. The class of the rule represents one of the classes of the domain, and not necessarily corresponds to the majority of the examples that satisfy the premises.

The proposed 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 (Spetch, 1990).

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* $\boldsymbol{\mu}_\mathbf{r} = (\mu_{r1}, \mu_{r2}, ..., \mu_{rd})^T$, where d is the dimensionality of the input space. This parameter is fixed during the learning (fitting) swtage;
- *two measures of dispersion*, $\boldsymbol{\sigma}_{\mathbf{r+}} = (\sigma_{r1+}, \sigma_{r2+}, ... \sigma_{rd+})^T$ and $\boldsymbol{\sigma}_{\mathbf{r-}} = (\sigma_{r1-}, \sigma_{r2-}, ... \sigma_{rd-})^T$. These are the free parameters adjusted during the learning stage.

It must be stressed that unlike the most common variations of radial basis functions, the parameters of center and dispersion are defined for *each* basis function and *each* attribute. In this case, the number of parameters is bigger. However, even using local measures of variability, the number of free parameters remains reasonably limited in the proposed model, because the vectors $\boldsymbol{\sigma}_{\mathbf{r+}}$ and $\boldsymbol{\sigma}_{\mathbf{r-}}$ are highly sparse: just a reduced subset of the whole set of domain variables is considered in each unit, as explained in Section 2.2, by initializing the network with a rule base. 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 $\boldsymbol{\sigma}_{\mathbf{r+}}$ and $\boldsymbol{\sigma}_{\mathbf{r-}}$: many of the variables are only bounded by an upper or bottom limit.

The activation function $a_r(\mathbf{x})$ used for the hidden unit r given an input $\mathbf{x}$ is a typical kernel function:

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

where $a_{rk}(x_k)$ is a function of attribute k at unit r. This function define axis-oriented kernel that neglect higher-order interaction among the variables at the advantage of requiring less parameters.

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

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

For unordered (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. Let $\mu_{rk,v}$ be the center of the kth variable at unit r for the corresponding value v. In an analogous way, we define $\sigma_{rk+,v}$. So, $a_{rk}(x_k)$, where $x_k = v$, is defined for unordered (symbolic) variables as:

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

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

[1] To simplify the notation, we still represent it by μ_{rk} instead of $\boldsymbol{\mu}_{\mathbf{rk}}$, as well as x_k instead of $\mathbf{x}_\mathbf{k}$.

built. The parameters of this final layer are not adapted by gradient learning: the weights w_r, $r \in \{1, ..., R\}$, 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 steps, one of the simplest measures is the *apparent accuracy*:

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

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.

In fact, for selecting rules during induction, this measure is not appropriate because a rule covering a single correct item have perfect accuracy, as discussed by Clark and Boswell (1991). The rule induction algorithms used in the experiments of Section 4. apply a measure that trades-off coverage and accuracy. For inference purposes, the apparent accuracy measure is quite robust, specially 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:

$$\text{Weighted accuracy}(r) = \frac{\sum_i 1(\mathrm{C}(\mathbf{x}_i) = \mathrm{C}(r))\, \mathrm{a}_r(\mathbf{x}_i)}{\sum_i \mathrm{a}_r(\mathbf{x}_i)} \quad (5)$$

where $1(\mathrm{C}(\mathbf{x_i}) = \mathrm{C}(r))$ is the indicator function for comparing the class of an example $\mathbf{x_i}$ and a unit r. It returns zero if $\mathbf{x}_i$ is not from the same class of r and so the numerator computes the influence only from examples $\mathbf{x}$ from the training set such that $\mathrm{C}(\mathbf{x}) = \mathrm{C}(r)$. The sum varies through all examples i.

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

$$y_j(\mathbf{x}) = \frac{\sum_r 1(\mathrm{C}(r) = j)\, w_r\, \mathrm{a}_r(\mathbf{x})}{\sum_r w_r\, \mathrm{a}_r(\mathbf{x})} \quad (6)$$

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

2.2 Initialization

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

Let $\mathbf{D_r}$ be the set of examples in the training set covered by the corresponding rule r, where it is considered only those examples that are of the same class of r. Unordered (symbolic) variables are represented in a 1-of-n encoding: $x_{rk,v}$ corresponds to the function $1(x_{rk} = v)$, where $1(x)$ is the indicator function, returning 1 if x is true and 0 otherwise; $v \in \{1, 2, ..., n\}$ represents the v-th possible value of x_k, and n is the dimensionality of its domain.

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 $\mathbf{D_r}$;
- in general, two spread parameters, σ_{rk+} and σ_{rk-}, are computed for each attribute k. The initialization is $\sigma_{rk+} = \sqrt{2}\,|\mathrm{b} - \mu_{rk}| \,/\, s$ and $\sigma_{rk-} = \sqrt{2}\,|\mu_{rk} - \mathrm{a}| \,/\, s$. 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 $P_{rk} = (X_{rk} < \mathrm{b})$ or $P_{rk} = (X_{rk} > \mathrm{a})$, k is used in the rule as an one-sided interval, so $\sigma_{rk-} = \infty$ or $\sigma_{rk+} = \infty$, for upper bounded or lower bounded intervals, respectively;

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

- the center parameters $\mu_{rk,v}$ are computed for each domain value of k. For all values v in the domain of 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:
  1. if $\mu_{rk,v} = 0$, $\sigma_{rk+,v} = \sqrt{2}/s$, $\sigma_{rk-,v} = \infty$. Typical values for s are 2 or 3;
  2. 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.

2.3 Probabilistic Interpretation

The initialization of the spread parameters is motivated by the normal probability distribution: nearly all of the data falls 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

[2] In the experiments described on Section 4, 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.

function $\exp(- (x - \mu^2) / 2\sigma^2)$ uses the dispersion parameter multiplied by 2. Note that the activation function is not identical to a normal density function: asymmetric dispersions are considered and the median is used as a measure of center. Using one-sided Gaussian functions to compute probabilities for categorical variables is not a common practice, but it was adopted for simplicity, where discrete variables are treated in a similar way to numerical ones.

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. This concentration can be computed by Equation (4). Rules induced by similar algorithms tends to correspond to the "bumps" of a multimodal distribution and so are more similar to kernel-based approaches for classification. Following Friedman and Fisher (1999), we call our framework *Bump Pruning and Fitting* (BPF).

As in the mixture of experts model (Jacobs et al., 1991), the BPF architecture can be interpreted as a combination of partial conditional probabilities $P(C \mid i, \mathbf{x})$, where i is the corresponding unit initialized from a rule, C is the class corresponding to unit i and $P(C \mid i, \mathbf{x})$ is simply the activation function $a_i(\mathbf{x})$ under the assumption of local variable independence. However, the mixing coefficients (5) are not a function of $\mathbf{x}$ itself and not even a partition of unity, as it would be necessary to model $P(C \mid \mathbf{x})$ according to the decomposition

$$P(C \mid \mathbf{x}) = \sum_{j=1}^{M} P(C \mid \mathbf{x}, j) P(j \mid \mathbf{x}) \tag{7}$$

This simplification may lead to less accurate predictions, but requires much less free parameters and facilitates creating an order criteria for pruning purposes, as described in Section 3. With the normalization, the output for each class C_k can be interpreted as $P(C_k \mid \mathbf{x})$ under the assumption that $P(j \mid \mathbf{x}) = P(j)$.

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 (Craven, 1996), and it is up to the modeler decide the appropriate trade-off.

Instead of following this path, we keep the rules under a probabilistic interpretation. Although at first sight those rules may seem less comprehensible than Boolean ones, in fact they may give more insight about the behavior of the data generation process by estimating the data distribution under each basis function. The data analyst uses a BPF-like approach if he or her decides that this representation is adequate for the problem to be modeled.

3. Pruning and Fitting

The central procedures of the BPF framework, as stated in Table 1, rely on methods for pruning and fitting parameters.

We adopted a straight pruning procedure that eliminates parameters following a pre-established order and iterates through this elimination while it is possible to keep pruning. Pruning is performed if classification accuracy does not decrease when discarding the respective parameter.

Initially, 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 the classification error. If the error does not increase (it may decrease), the unit is permanently removed.

After dealing with units, the next pruning stage concentrates on the individual premises of each rule. The parameters defining the premises are the spread parameters $\boldsymbol{\sigma}$. The order of evaluation is defined by a simple sensitivity analysis criteria.

Let w_σ be the output weight of a given rule/unit when setting one of its given spread parameters σ to infinite, which corresponds to ignoring the parameter. If w is the original weight of the rule, then the equation

$$\text{EvaluateSigma}(\sigma) = (w_\sigma - w) / w \tag{8}$$

can be interpreted as a proportional gain on accuracy of an individual rule when pruning σ. All spread parameters are queued in a decreasing order of EvaluateSigma(σ) and processed in an analogous way to unit pruning. This two-stage pruning procedure is repeated while the last cycle results in some simplification.

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 (1998) 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 pruning stage can also corrects some errors introduced by the gradient-based optimization. The weights in the output layer are adjusted after each update of the parameter vector. This is a variation of a generalized expectation-maximization algorithm.

The purpose of the fitting procedure is correcting some initial errors introduced by the initialization procedure (which is natural to occur, since we are dealing with different knowledge representations) as well as redistributing all the kernels at the same time with respect to the MSE criteria. This is a complement to the learning of rules by separate-and-conquer covering, which evaluates only one rule at a time. The fitting procedure may result in extra redundant components that will lead to further pruning. In the reported experiments, no more than three cycles of pruning and fitting were performed in any data set.

4. Experiments

Some experiments were performed to evaluate the merits of BPF, as well as to provide more evidence that could be used to explain the effectiveness of this framework. Lim, Loh and Shih (1999, in press) performed a comparison of different algorithms in a benchmark of classification data sets, most from the UCI Machine Learning Repository (Blake, Keogh & Merz, 1998). Fifteen of these data sets were used in the experiments of this section. Table 2 gives a brief description of them.

Table 2. Brief description of the data sets. The first column gives the identifiers that will be used to reference these data sets in the text. The last two columns informs the number of training and test instances, respectively. The label "CV" stands for 10-fold cross-validation using the training set.

ID	DESCRIPTION	#TRAIN	#TEST
`bcw`	Wisconsin breast cancer	683	CV
`bld`	BUPA live disorders	345	CV
`bos`	Boston housing	506	CV
`cmc`	Contraceptive method choice	1473	CV
`hea`	StatLog heart disease	270	CV
`led`	LED display	2000	4000
`pid`	PIMA indian diabetes	532	CV
`sat`	StatLog satellite image	4435	2000
`seg`	Image Segmentation	2310	CV
`smo`	Attitude towards smoking	1855	1000
`tae`	Teaching assistant evaluation	151	CV
`thy`	Thyroid disease	3772	3428
`veh`	StatLog vehicle silhouette	846	CV
`vot`	Congressional voting records	435	CV
`wav`	Waveform	600	3000

One of the rule induction algorithms used in the experiments was the Patient Rule Induction Method (PRIM) developed by Friedman and Fisher (1999)[3]. PRIM has two characteristics that makes it suitable for these experiments: its search bias is a typical recursive covering algorithm that identifies bumps of the data distribution, which in principle generates rule sets more similar to the concept of mixture models; also, it allows multiple tests for symbolic attributes in a single premise. Other algorithm, C4.5RULES (Quinlan, 1993), was also applied. Table 3 shows results from rule sets induced by PRIM and C4.5RULES that were post-pruned by the same pruning algorithm described on Section 4. The rule weights for these variations are computed according to Equation (5). BPF-P and BPF-C are BPF models initialized by PRIM and C4.5RULES, respectively.

The metric of model complexity was the number of premises in a rule set. 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 trouble with `tae`. The average ratio of simplification[4] 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 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 following this approach usually results in smaller rule sets than the approach 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] An implementation is available at `http://www-stat.stanford.edu/~jhf/SuperGEM.html`.

[4] This ratio is defined as the number of parameters of BPF divided by the number of parameters of the corresponding PRIM (or C4.5RULES) pruned rule set. The data sets `cmc` and `smo` were treated as outliers and excluded from the computation of this statistic.

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 Accuracy								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	–

BPF models were trained with a conjugate gradient algorithm till convergence. The training set was used as the pruning set which suggests that high simplifications can be performed without relying in extra data sources. Using independent pruning sets results in proportional gains. Table 4 shows the results obtained with the largest data sets[5] where one third of the original training data set was used for pruning only. PRIM was used as the initialization algorithm for the reported BPF-P results. It must be observed that the accuracy gains are in fact due the combination of kernels, since PRIM with rules combined in a similar fashion to (6) achieve similar results.

This benchmark can be used to provide evidences about the causes of the effectiveness of the BPF framework. After the initial remapping, one can observe that the training set error is usually higher than in the original Boolean rule set. However, the generalization error suffers less impact. BPF acts as a filter that smoothes the original representation. The parameter s described in Section 2.2 is a global parameter for controlling the initial smoothness. The amount of initial smoothness is an issue of bias/variance trade-off (Geman, Bienenstock & Doursat, 1992). It may happen that the pruning and fitting cycles are not able to satisfactorily correct an excessive initial smoothing, detected by a low generalization error in cross-validation or independent sample setups. If one does not want to use cross-validation or an independent sample to detect this problem, it may observe if the training error is unusually high, when compared to the original rule base. In all the reported experiments of Table 2 and Table 3, we used $s = 2$ without relying in model selection procedures to refine this value.

Table 4. Comparison of BPF-P and PRIM in the largest data sets. An independent pruning set corresponding to one third of the original training set was used. BPF-P still maintains a fair difference.

Set	PRIM		BPF-P	
	complexity	error	complexity	error
`led`	86	0.265	**58**	0.277
`sat`	184	0.161	**104**	0.134
`wav`	34	0.251	**7**	0.226

After convergence, BPF train errors are considerably lower than their initial values. The generalization error may improve, but it is hardly better than the original rule base. The bias introduced by the rule base seems to be strong enough to avoid significant improvements, specially considering that the same training set is used in the gradient-based learning.

[5] `smo` was discarded because it converged to the default rule of classification, while the original thy was already resulting in a very reduced rule set.

Table 5. Comparison of BPF-P and PRIM with respect to the initialization error in both the training set and test set error. Notice that the test errors are relatively similar, while train errors present a stronger variation.

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

5. Related Work

Approaches for initialization of neural-like learning models from other sources of knowledge date back at least from the initial work on knowledge-based neural networks (KBANNs) by Towell and Shavlik (1989). The KBANN mapping includes the treatment of layers of rules, i.e., where rules are used to predict inputs for other rules.

The principles underlying BPF methodology relies more on the equivalence of kernels and propositional rules than in the graph properties of rule bases. Andrews and Geva (1996) provides a more extensive discussion and a survey of some hybrid systems of rules and neural networks under the point of view of local basis functions.

There is a wealthy literature involving local basis functions and rule-based knowledge, which is in fact one of the main principles of fuzzy logic. Kosko (1997) provides extra topics, including the treatment of oblique rules, i.e., rules defined by local basis functions that are not axis-oriented.

Among the many hybrids of local basis functions and propositional rules, Tresp, Hollatz and Ahmad (1997) present one of the most similar frameworks when compared to BPF. However, few empirical results are discussed. The networks of Trep et al. model joint density functions instead of the conditional probability functions of BPF. These networks also provide a different interpretation for rule premises, as well as a different initialization procedure. For instance, premises concerning numerical variables represent single points with a measure of variability, i.e., instead of modeling the original propositional interval (and so allowing one-sided constraints), those premises are of the form `variable = value`, with a uncertainty interval represented by normality assumption and a standard deviation parameter.

RIPPER (Cohen, 1995) is a separate-and-conquer rule learning algorithm that also performs cycles of pruning and rule modification. However, the criteria of rule modification differs from BPF by applying the same concepts of separate-and-conquer used to rule induction. There is no rule smoothing, which was one of the main factors that contributed for model simplification under BPF framework. It is important to stress that RIPPER is intended to be a complete rule induction algorithm, while BPF is only a framework over rule inducers. There is no reason for not to use RIPPER induced rules with BPF.

One of the main ideas of BPF, fitting local measures of dispersion, is surveyed by Wettschereck, Aha and Mohri (1997) under the point of view of feature weighting for lazy learning algorithms, but that can be easily extended for kernel-based models.

Using rule and tree induction as methods of feature selection for other learning systems has many applications, such as joint density estimation by instance-based learning (Smyth, Gray & Fayyad, 1995). Under the point of view of neural network learning, Kubat (1998) uses partitions defined by a tree to initialize a radial basis function network and then performing learning of the (linear) output layer as a fast way to get improved classification performance.

Notice that these approaches focus in increasing prediction accuracy. BPF hardly improves accuracy over the original representations, at least in the reported experiments. One of the reasons are the topology limitations imposed in this architecture to help comprehensibility of the model. For instance, Kubat's mapping methodology fully connects the kernels (originated from the regions defined by the original tree) with the output units of the neural network. This helps getting improved generalization, but may harm comprehensibility, since rules related to specific classes are used to compute results for all classes.

Gray (1997) points that many hybrid systems of fuzzy logic and neural networks do not live up to the expectation of real gains in predictive power when compared to the initial rule base.

Finally, an early version of this framework was discussed by Silva and Ludermir (1999).

6. Conclusion

The BPF framework does not intend to offer new approaches for rule induction, but rather building on existing ones by allowing alternatives 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.

BPF follows the principle of extracting comprehensible models from more complex ones without concerns about increasing predictive power. As pointed out by Domingos (1999), this is a promising approach to achieve a desirable trade-off between accuracy and complexity. An interesting future work is letting the pruning stage trades-off some accuracy for extra simplification and measuring how well BPF would recover this lost accuracy during the fitting step.

A relative limitation of BPF is relying in gradient-based optimization that is typically much slower than greedy separate-and-conquer rule induction algorithms. However, it may be faster than traditional neural networks, since the starting point is closer to a local minimum. The proximity of a local minimum may be a disadvantage for hybrid systems of neural networks and rules, if little gain in predictive power is expected. This limitation does not directly apply for BPF, since we assume that no significant accuracy gains will result from this process.

Acknowledgements

The authors would like to thank the anonymous reviewers for their suggestions.

This work was supported by a CNPq grant.

References

Andrews, R., & Geva, S. (1996). Rules and local function networks. *Proceedings of the Rule Extraction from Trained Artificial Neural Networks Workshop* (pp. 1-15). University of Sussex, Brighton, UK.

Blake, C., Keogh, E., & Merz, C. J. (1998). *UCI Repository of machine learning databases.* Irvine, CA: University of California, Department of Information and Computer Science. URL: `http://www.ics.uci.edu/~mlearn/MLRepository.html`.

Cherkassky, V., & Mulier, F. (1998). *Learning from data: Concepts, theory and methods*. New York: Wiley Interscience.

Clark, P., & Boswell, R. (1991). Rule induction with CN2: some recent improvements. *Proceedings of the 5th European Conference on Machine Learning* (pp. 151-163). Springer Verlag.

Cohen, W. W. (1995). Fast effective rule induction. *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 115-123). San Francisco: Morgan Kaufmann.

Craven, M. W. (1996). *Extracting comprehensible models from trained neural networks.* Doctoral dissertation, Computer Sciences Department, University of Wisconsin, Madison.

Domingos, P. (1999) The role of Occam's Razor in knowledge discovery. *Data Mining and Knowledge Discovery, 3,* 1-19.

Fayyad, U. M., Piatetsky-Shapiro, G., & Smyth, P. (1996) From data mining to knowledge discovery: an overview. In U. Fayyad, G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (Eds.), *Advances in Knowledge Discovery and Data Mining,* 1-34. AAAI/MIT Press.

Friedman, J. H., & Fisher, N. I. (1999). Bump hunting in high-dimensional data. *Statistics and Computing, 9,* 123-143.

Fürnkranz, J. (1999). Separate-and-conquer rule learning. *Artificial Intelligence Review, 13*, 3-54.

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural networks and the bias/variance dilemma. *Neural Computation, 4*, 1-58.

Gray, A. (1997) *Hybrid systems FAQ*. URL: `http://divcom.otago.ac.nz:800/COM/INFOSCI/SMRL/people/andrew/publications/faq/hybrid/parts/Info.htm`

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., & Hinton, G. E. (1991). Adaptive mixture of local experts. *Neural Computation, 3*, 79-87.

Kosko, B. (1997). *Fuzzy enginnering*. Prentice Hall.

Lim, T., Loh, W., & Shih, Y. (1999). A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms. *Machine Learning Journal* (in press).

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. San Mateo, CA: Morgan Kaufmann.

Silva, R. B. A, & Ludermir, T. B. (1999). Neural network methods for rule induction. *Proceedings of the International Joint Conference on Neural Networks 1999*, paper #2099.

Smyth, P., Gray, A., & Fayyad, U. M. (1995). Retrofitting decision tree classifiers using kernel density estimation. *Proceedings of the Twelfth International Conference on Machine Learning* (pp. 506-514). San Francisco: Morgan Kaufmann.

Specht, D. (1990). Probabilistic neural networks. *Neural Networks, 3*, 109-118.

Towell, G., & Shavlik, J. (1989). An approach to combing explanation-based and neural learning algorithms. *Connection Science, 1,* 233-255.

Tresp, V., Hollatz, J., & Ahmad. S. (1997). Representing probabilistic rules with networks of gaussian basis functions. *Machine Learning, 9*, 1-31.

Wettschereck, D., Aha, D. W., & Mohri, T. (1997). A review and comparative evaluation of feature weighting methods for lazy learning algorithms. *Artificial Intelligence Review, 11*, 273-314.