Beyond Sequential Covering – Boosted Decision Rules

Krzysztof Dembczyński[1], Wojciech Kotłowski[1], and Roman Słowiński[2]

[1] Poznań University of Technology, 60-965 Poznań, Poland
kdembczynski@cs.put.poznan.pl, wkotlowski@cs.put.poznan.pl
[2] Poznań University of Technology, 60-965 Poznań, Poland,
and Systems Research Institute, Polish Academy of Sciences, 01-447 Warsaw, Poland
rslowinski@cs.put.poznan.pl

Abstract. From the beginning of machine learning, rule induction has been regarded as one of the most important issues in this research area. One of the first rule induction algorithms was AQ introduced by Michalski in early 80's. AQ, as well as several other well-known algorithms, such as CN2 and Ripper, are all based on sequential covering. With the advancement of machine learning, some new techniques based on statistical learning were introduced. One of them, called boosting, or forward stagewise additive modeling, is a general induction procedure which appeared to be particularly efficient in binary classification and regression tasks. When boosting is applied to induction of decision rules, it can be treated as generalization of sequential covering, because it approximates the solution of the prediction task by sequentially adding new rules to the ensemble without adjusting those that have already entered the ensemble. Each rule is fitted by concentrating on examples which were the hardest to classify correctly by the rules already present in the ensemble. In this paper, we present a general scheme for learning an ensemble of decision rules in a boosting framework, using different loss functions and minimization techniques. This scheme, called ENDER, is covered by such algorithms as SLIPPER, LRI and MLRules. A computational experiment compares these algorithms on benchmark data.

1 Introduction

We consider the classification problem that consists in assigning objects described by several attributes to pre-defined decision classes. Using a set of training examples, our task is to build a classifier that assigns new objects to decision classes as accurately as possible. There are several learning algorithms introduced for this task. In the following, we consider classifiers based on decision rules.

Decision rule is a logical statement of the form: *if* [*condition*], *then* [*decision*]. If an object satisfies the condition part of the rule, then the decision is taken; otherwise no action is performed. A rule can be treated as a simple classifier that gives a constant response for the objects satisfying the conditions, and abstains from the response for all other objects. The main advantage of decision rules is their simplicity and human-readable form.

Induction of decision rules has been widely considered in the early machine learning approaches [1, 2, 3, 4]. The most popular algorithms were based on a *sequential covering* procedure (also known as separate-and-conquer approach). According to [4], this

J. Koronacki et al. (Eds.): Advances in Machine Learning I, SCI 262, pp. 209–225.
springerlink.com

procedure can be described as follows: "learn a rule that covers a part of the given training examples, remove the covered examples from the training set (the separate part) and recursively learn another rule that covers some of the remaining examples (the conquer part) until no examples remain".

Apart from the sequential covering, some other approaches to rule induction exist. For instance, the apriori-based algorithms are also used for induction of predictive rules [5, 6]. There are several rule-based approaches of lazy learning type [7, 8], possibly combined with instance-based methods [9, 10]. Other algorithms are based on Boolean reasoning and mathematical programming [11]. Let us also notice that decision rule models are strongly associated with rough set approaches to knowledge discovery [12, 13, 14, 15, 16, 17].

Recently, some new algorithms have been introduced that follow another approach to rule induction. They treat decision rules as subsidiary, base classifiers in the ensemble. More precisely, decision rule classifiers are built using boosting [18] or forward stagewise additive modeling [19]. The examples of such algorithms are SLIPPER [20], LRI [21], RuleFit [22], and MLRules [23]. We will refer to this family of algorithms as *boosted decision rules*.

In this paper, we discuss the relation between sequential covering and boosted decision rules. As we will see, these two procedures are quite similar and the latter can be seen as a generalization of the former. We will show a general scheme for rule induction that we called ENDER. Different rule learning algorithms fall into this scheme as special cases.

The paper is organized as follows. Section 2 states formally the problem of classification. In Section 3, the decision rule is defined. Sequential covering is briefly described in Section 4. We show how the boosted decision rules are related to sequential covering in Section 5. Two instances of boosting algorithms for rule induction, SLIPPER and MLRules, are also described there. Several extensions of boosted decision rules are discussed in Section 6. Experimental comparison of different rule induction algorithms is given in Section 7. The last Section concludes the paper.

2 Classification Problem

We consider the classification problem in which the aim is to predict for an object an unknown class label of attribute y taken from set $\{1,\ldots,K\}$, knowing joint values of attributes $\mathbf{x} = (x_1, x_2, \ldots, x_n) \in \mathcal{X}$. The task is to find function $f(\mathbf{x})$, called *classifier*, that predicts accurately the value of y. The accuracy of a single prediction is measured by *loss function* $L(y, f(\mathbf{x}))$ which determines the penalty for predicting $f(\mathbf{x})$ when the actual value is y. The overall accuracy of function $f(\mathbf{x})$ is measured by the expected loss, called *risk*, over joint distribution $P(y, \mathbf{x})$:

$$R(f) = \mathbb{E}_{y\mathbf{x}} L(y, f(\mathbf{x})).$$

Therefore, the optimal risk-minimizing classification function, called *Bayes classifier*, is given by:

$$f^* = \arg\min_f \mathbb{E}_{y\mathbf{x}} L(y, f(\mathbf{x})).$$

Since $P(y,\mathbf{x})$ is generally unknown, the learning procedure uses only a set of training examples $\{y_i,\mathbf{x}_i\}_1^N$ to construct f to be the best possible approximation of f^*. Usually, this is performed by minimization of the *empirical risk*:

$$R_{emp}(f)=\frac{1}{N}\sum_{i=1}^{N}L(y_i,f(\mathbf{x}_i)),$$

where f is chosen from a restricted family of functions.

The most natural loss function is the *0-1 loss*:

$$L_{0-1}(y,f(\mathbf{x}))=\begin{cases}0, & \text{if } y=f(\mathbf{x}),\\ 1, & \text{if } y\neq f(\mathbf{x}).\end{cases} \tag{1}$$

The expected value of this loss function is simply a misclassification error of $f(\mathbf{x})$ defined by $\Pr(y\neq f(\mathbf{x}))$. Thus, Bayes classifier has the following form:

$$f^*(\mathbf{x})=\arg\min_{f(\mathbf{x})}\mathbb{E}_{y|\mathbf{x}}L_{0-1}(y,f(\mathbf{x}))=\arg\max_{k\in\{1,\ldots,K\}}\Pr(y=k|\mathbf{x}). \tag{2}$$

By minimizing the 0-1 loss function, we estimate K regions in the attribute space, such that in the k-th region, the k-th class is observed with the highest probability.

For the sake of simplicity, we will confine our considerations to the two-class problem, referred to as *binary classification*. Let us assume that $y\in\{-1,1\}$ and the classifier gives continuous responses, $f(\mathbf{x})\in\mathbb{R}$. In this case, the loss function can be expressed by $L(yf(\mathbf{x}))$, where $yf(\mathbf{x})$ is called *margin*. The positive margin means no error, and its magnitude tells, intuitively, what is the credibility of assigning an object to a given class. The *margin 0-1 loss* function is therefore:

$$L_{0-1}(yf(\mathbf{x}))=[\![yf(\mathbf{x})<0]\!],$$

where $[\![\pi]\!]$ is the Boolean test equal to 1 if predicate π is true, and 0 otherwise.

3 Decision Rule

Decision rule is a simple logical pattern having the form:

"if [*condition*] then [*decision*]".

Its main advantage is simplicity and human-interpretable form that can model interactions between several attributes for which the condition is defined. As an example of a decision rule, let us consider a rule generated from the well-know CREDIT-G benchmark data set concerning the problem of predicting a risk-level of bank customers:

if CHECKING STATUS $=$ *no checking account*
and OTHER PAYMENT PLANS $\neq$ *bank*
and AGE $\in[22.5,66.5]$
and CREDIT AMOUNT ≤ 10924.5,
then CUSTOMER $=$ *good*.

From a machine learning perspective, such a rule can be treated as a simple classifier that gives a constant response to examples satisfying the condition part, and abstains from the response for all other examples. Formally, a decision rule can be defined as follows.

Let $X_j \subset \mathscr{X}$ be a domain of attribute $j \in \{1,\ldots,n\}$. The condition part of the rule consists of a conjunction of elementary expressions of the general form

$$x_j \in S_j, \tag{3}$$

where x_j is the value of object $\mathbf{x}$ on attribute j and S_j is a subset of X_j. In particular, elementary expressions are of the form $x_j \geq s_j$, $x_j \leq s_j$, for quantitative attributes, and $x_j = s_j$, $x_j \neq s_j$, for qualitative attributes, where s_j is taken from a domain of attribute j. Let Φ be the set of elementary expressions constituting the condition part of the rule, and let $\Phi(\mathbf{x})$ be a function indicating whether object $\mathbf{x}$ satisfies the conjunction of elementary expressions Φ. In other words, $\Phi(\mathbf{x})$ defines an arbitrary axis-parallel region in the attribute space. We say that a rule *covers* object $\mathbf{x}$ if it belongs to this region, i.e. $\Phi(\mathbf{x}) = 1$. The number of training examples covered by the rule is referred to as *rule coverage*. The decision, or response, denoted by α, is a real value assigned to the region defined by Φ. Therefore, we define a decision rule as:

$$r(\mathbf{x}) = \alpha\Phi(\mathbf{x}).$$

Note that the decision rule takes only two values, $r(\mathbf{x}) \in \{\alpha, 0\}$, depending whether $\mathbf{x}$ satisfies the condition part or not.

Having defined the decision rule, we now need an algorithm that induces such rules from a set of training examples and combines them into a powerful classifier.

4 Sequential Covering

Almost all early algorithms for rule learning were based on sequential covering. The most popular are AQ [1], CN2 [2], Ripper [3], and LEM [14]. Sequential covering relies on learning a rule in each step, that covers a part of a given set of training examples, removing the covered examples from the training set, and repeating this step until no example remains. This procedure is repeated separately for each class. In each turn, the rules cover examples from one class only. These examples are called *positive*, while the others, *negative*.

This approach is more formally presented as Algorithm 4.1. It starts with an empty set of decision rules and successively adds rules to it, until all positive examples are covered. A single rule is built by the FindBestRule procedure in order to cover positive examples only. This can be done by specializing the current rule by adding new elementary expressions to its condition part. In order to move towards the goal of finding a rule that covers no negative examples, the algorithm selects the elementary expressions that optimize the purity measure of the rule. The purity measure is a function of the numbers of positive and negative examples. For instance, this can be the fraction of positive examples in the set of covered examples.

This approach to rule induction encounters several problems. For example, the set of decision rules does not usually cover the entire attribute space. Moreover, several

Algorithm 4.1. Sequential covering

input : set of training examples $\{y_i, \mathbf{x}_i\}_1^N$,
k – label of positive class.
output: set of decision rules R_k for k-th class.
$R_k = \emptyset$
$Pos = \{\mathbf{x}_i : y_i = k, i = 1, \ldots, N\}$ // positive examples
$Neg = \{\mathbf{x}_i : y_i \neq k, i = 1, \ldots, N\}$ // negative examples
$m = 0$
while $Pos \neq \emptyset$ **do**
  $r_m(\mathbf{x}) = \text{FindBestRule}(Pos, Neg)$
  $Pos = Pos - \{\mathbf{x}_i : r_m(\mathbf{x}) = 1\}$ // remove covered examples
  $R_k = R_k \cup r_m(\mathbf{x})$
  $m = m + 1$
end

rules may overlap. Thus, there is a problem how to classify new examples by a set of rules. In many rule induction algorithms, the rule generation phase and the classification phase are considered separately. First, the algorithm generates a set of rules. Next, one of several possible strategies is used to classify new examples. Usually, different voting schemes are considered. The weights of the rules used in the voting are simple statistics computed over examples covered by the rules. A popular choice is a relative number of covered examples or an estimate of conditional class probability within the rule. The latter suggests that the rules are treated as independent ones, what is not true taking into account how the sequential covering procedure works.

5 Boosted Decision Rules

The problem described above can be easily solved, if we first assume some form of the ultimate rule-based classifier. Let the classification function be a linear combination of M rules:

$$f_M(\mathbf{x}) = \alpha_0 + \sum_{m=1}^{M} r_m(\mathbf{x}), \tag{4}$$

where α_0 is a constant value, which can be interpreted as a default rule, covering the whole attribute space. This is an ensemble of simple and interpretable base classifiers. Single decision rule $r_m(\mathbf{x})$ defines an increase or decrease of $f_M(\mathbf{x})$ by α_m if a condition $\Phi_m(\mathbf{x})$ is met. In other words, this is a weighted majority voting. The prediction is made by $\text{sgn}(f_M(\mathbf{x}))$.

Having such a form of the classifier, we can proceed in a similar way to sequential covering. We add the rules one by one in order to minimize the empirical risk. In each subsequent iteration, a new rule is added by taking into account previously generated rules. Let $f_{m-1}(\mathbf{x})$ be a classification function after $m-1$ iterations, consisting of the first $m-1$ rules and the default rule. In the m-th iteration, a decision rule can be obtained by solving:

$$r_m(\mathbf{x}) = \arg\min_{r(\mathbf{x})} R_{emp}(f_{m-1}(\mathbf{x}) + r(\mathbf{x})) = \arg\min_{\Phi,\alpha} \sum_{i=1}^{N} L(y_i, f_{m-1}(\mathbf{x}_i) + \alpha\Phi(\mathbf{x}_i)). \tag{5}$$

This procedure corresponds to sequential covering, since the value of the margin 0-1 loss function decreases down to 0 for all correctly covered training examples, and there is no need for another rule to cover them again. This corresponds to removing such objects from the training set. All the rules will obtain the same absolute value of rule response $|\alpha|$, and the sign decides for which class the rule votes. The classification is then a simple majority voting.

Instead of the margin 0-1 loss, one can use, however, another loss function, like exponential loss or logit loss. These are surrogates (upper-bounding the 0-1 loss) commonly used in classification tasks. These functions are convex and differentiable, which makes the minimization process easier to cope with. The *exponential loss* is defined as:

$$L_{\exp}(yf(\mathbf{x})) = \exp(-yf(\mathbf{x})). \tag{6}$$

This loss function is used in AdaBoost [18]. The *logit loss*

$$L_{\log}(yf(\mathbf{x})) = \log(1+\exp(-2yf(\mathbf{x}))) \tag{7}$$

is commonly used in statistics. These two loss functions are sensitive to the value of $yf(\mathbf{x})$. The Bayes classifier for both of them has the form [19]:

$$f^*(\mathbf{x}) = \frac{1}{2}\log\frac{\Pr(y=1|\mathbf{x})}{\Pr(y=-1|\mathbf{x})}, \tag{8}$$

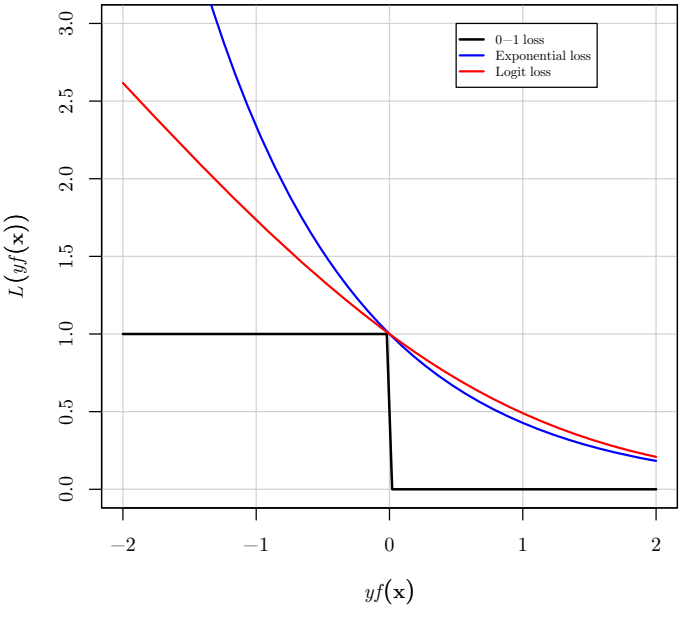

Fig. 1. Loss functions $L(yf(\mathbf{x}))$ expressed through the margin $yf(\mathbf{x})$ for binary classification

Algorithm 5.1. Ensemble of decision rules – ENDER

input : set of training examples $\{y_i, \mathbf{x}_i\}_1^N$,
$L(y, f(\mathbf{x}))$ – loss function,
M – number of decision rules to be generated.
output: ensemble of decision rules $f_M(\mathbf{x})$.

$\alpha_0 = \arg\min_\alpha \sum_{i=1}^N L(y_i, \alpha)$
$f_0(\mathbf{x}) = \alpha_0$;
for $m = 1$ *to* M **do**
  $\Phi_m = \arg\min_\Phi \mathscr{L}_m(\Phi)$
  $\alpha_m = \arg\min_\alpha \sum_{i=1}^N L(y_i, f_{m-1}(\mathbf{x}_i) + \alpha\Phi_m(\mathbf{x}_i))$
  $r_m(\mathbf{x}) = \alpha_m\Phi_m(\mathbf{x})$
end

which is the logit transform of the conditional probabilities. Expression (8) can be inverted to give:

$$\Pr(y = 1|\mathbf{x}) = \frac{1}{1 + \exp(-2f^*(\mathbf{x}))}. \tag{9}$$

Therefore, minimization of these loss functions on the training set can be seen as estimation of conditional probabilities $\Pr(y = 1|\mathbf{x})$. The sign of $f(\mathbf{x})$ estimates in turn the class with a higher probability. Characteristics of these loss functions are shown in Figure 1.

Using these loss functions in (5) results in the boosting-like algorithm for rule induction, we presented in [24] and called ENDER. We outline its main steps in Algorithm 5.1. First, the default rule is computed:

$$\alpha_0 = \arg\min_\alpha \sum_{i=1}^N L(y_i, \alpha). \tag{10}$$

In each next iteration, we proceed in two steps:

1. Find Φ_m by minimizing a functional $\mathscr{L}_m(\Phi)$ derived from (5) that does not depend of α:
$$\Phi_m = \arg\min_\Phi \mathscr{L}_m(\Phi). \tag{11}$$
2. Find α_m, the solution to the following line-search problem:
$$\alpha_m = \arg\min_\alpha \sum_{i=1}^N L(y_i, f_{m-1}(\mathbf{x}_i) + \alpha\Phi_m(\mathbf{x}_i)). \tag{12}$$

The algorithm finds iteratively the best rules minimizing the empirical risk. $\mathscr{L}(\Phi)$, which denotes a purity measure, is greedily minimized by a heuristic that builds the condition part of the decision rule. It works in a similar way to the FindBestRule procedure. The difference is that it does not stop when no negative example is covered, but when $\mathscr{L}_m(\Phi_m)$ cannot be decreased. The response of the rule is the weight in a voting procedure. It is computed by taking into account the previously generated rules, thus

the dependencies between rules are not ignored. The entire procedure is very intuitive and, formally, well-founded. Below, we will state that several rule learning algorithms fall into this ENDER scheme.

In the case of the exponential loss, the ENDER algorithm works as follows. The response of the decision rule is computed according to:

$$\alpha_m = \arg\min_{\alpha} \sum_{i=1}^{N} e^{-y_i(f_{m-1}(\mathbf{x}_i)+\alpha\Phi_m(\mathbf{x}_i))} = \frac{1}{2}\log\frac{\sum_{y_i=1}\Phi(\mathbf{x}_i)e^{-f_{m-1}(\mathbf{x}_i)}}{\sum_{y_i=-1}\Phi(\mathbf{x}_i)e^{f_{m-1}(\mathbf{x}_i)}}. \tag{13}$$

The solution of α_m can be put to (5), and after straightforward transformations, we get the following form of the purity measure:

$$\mathscr{L}_m(\Phi) = \sqrt{\left(\sum_{y_i=1}\Phi(\mathbf{x}_i)w_i^{(m)}\right)} + \sqrt{\left(\sum_{y_i=-1}\Phi(\mathbf{x}_i)w_i^{(m)}\right)}, \tag{14}$$

where $w_i^{(m)} = e^{-y_i f_{m-1}(\mathbf{x}_i)}$ can be treated as a weight of the i-th training example in the m-th iteration. This is also how the SLIPPER algorithm works [20].

In the MLRules algorithm [23], which is the next to describe, we derive rules from the maximum likelihood principle that can be seen as minimization of the logit loss (7) by the so-called gradient descent technique.

In case of the logit loss, there is no analytical solution to (12). To speed up the computations, instead of numerically solving the line search problem, we perform a single Newton-Raphson step, similarly as in [25]:

$$\alpha_m = -\left.\frac{\sum_{\Phi(\mathbf{x}_i)=1}\frac{\partial}{\partial\alpha}L_{\log}(y_i(f_{m-1}(\mathbf{x}_i)+\alpha\Phi_m(\mathbf{x}_i)))}{\sum_{\Phi(\mathbf{x}_i)=1}\frac{\partial^2}{\partial\alpha^2}L_{\log}(y_i(f_{m-1}(\mathbf{x}_i)+\alpha\Phi_m(\mathbf{x}_i)))}\right|_{\alpha=0}. \tag{15}$$

From (5) we derive the form of $\mathscr{L}_m(\Phi)$, in such a way that $\mathscr{L}_m(\Phi)$ does not depend on α. This can be done by approximating (5) up to the first order with respect to α:

$$r_m(\mathbf{x}) \simeq \arg\min_{\Phi,\alpha} \sum_{i=1}^{N}\left(L(y_i, f_{m-1}(\mathbf{x}_i)) - \alpha\Phi(\mathbf{x}_i)\tilde{w}_i^{(m)}\right), \tag{16}$$

where

$$\tilde{w}_i^{(m)} = -\left.\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\right|_{f(\mathbf{x}_i)=f_{m-1}(\mathbf{x}_i)}.$$

Since $\tilde{w}_i^{(m)}$, $i = 1,\ldots,N$, is the negative gradient of the function to be minimized, this technique is called *gradient descent*. It is easy to see that the optimal solution with respect to Φ is obtained by minimizing:

$$\mathscr{L}_m(\Phi) = -\sum_{\Phi(\mathbf{x}_i)=1}\alpha\tilde{w}_i^{(m)}, \tag{17}$$

since $\sum_{i=1}^{N} L(y_i, f_{m-1}(\mathbf{x}_i))$ is constant at a given iteration, and thus it does not change the solution. Observe that for a given value of α, the solution only depends on

$\sum_{\Phi(\mathbf{x}_i)=1} \tilde{w}_i^{(m)}$, so the minimization of (17) can be finally reformulated to the minimization of the following term:

$$\mathscr{L}_m(\Phi) = -\left| \sum_{\Phi(\mathbf{x}_i)=1} \tilde{w}_i^{(m)} \right|, \tag{18}$$

because the sign and the magnitude of α may be established afterwards.

Another boosted rule induction algorithm, LRI [21], uses a specific reweighting schema (cumulative error), similar to Breiman's Arc-xf algorithm [26]. For the two-class problem, however, this method can also be explained in terms of the loss function minimization, as it was done in [27] for Arc-xf. It follows that LRI minimizes a polynomial loss function by the gradient descent technique. Let us also point out that a single rule in LRI is a bit more complex in comparison with those generated by other algorithms, because the rule is a DNF-formula, i.e., disjunction of conjunctions of elementary expressions, instead of a single conjunction.

6 Extensions

6.1 Rule Coverage

Several problems associated with a typical sequential covering approach have been pointed out in [28]. One of the problems concerns the discrimination and completeness of the rule. In other words, "how good are the examples from positive class separated from negative examples, and how many positive examples are covered?" The usual way of answering this question consists in using different purity measures and their combinations, sometimes defined ad hoc, to control the process of constructing a single decision rule. In the approach followed by ENDER, the answer is very natural. In each iteration, where we find a rule which is minimizing the empirical risk, we can slightly modify the minimization technique in order to control the trade-off between discrimination and completeness.

Let us restrict α in (5) to $\alpha \in \{-\beta, \beta\}$, where β is a parameter of the algorithm. Then, (5) becomes:

$$r_m(\mathbf{x}) = \arg\min_{\Phi,\pm\beta} \left(\sum_{\Phi(\mathbf{x}_i)=1} L(y_i, f_{m-1}(\mathbf{x}_i) \pm \beta) + \sum_{\Phi(\mathbf{x}_i)=0} L(y_i, f_{m-1}(\mathbf{x}_i)) \right). \tag{19}$$

Since β is fixed, we refer this technique to as *constant-step minimization*. In the case of the exponential loss, we can prove the following theorem.

Theorem 1. *[29] The problem of solving (19) for exponential loss (6) and step length β is equivalent to minimization of*

$$\mathscr{L}_m(\Phi) = \sum_{i \in R_-} w_i^{(m)} + \ell \sum_{\Phi(\mathbf{x}_i)=0} w_i^{(m)}, \tag{20}$$

where $R_- = \{i : y_i \alpha \Phi(\mathbf{x}_i) < 0\}$ indicates the training examples misclassified by the rule, and:

$$w_i^{(m)} = e^{-y_i f_{m-1}(\mathbf{x}_i)}, \qquad \ell = \frac{1 - e^{-\beta}}{e^{\beta} - e^{-\beta}}, \qquad \beta = \log \frac{1-\ell}{\ell}.$$

Expression (20) has a nice interpretation. The first term corresponds to examples "misclassified" by the rule, while the second term – to examples which are not classified by the rule at all. Value l plays the role of a penalty for abstaining from classification and establishes a trade-off between not classified and misclassified examples. It is easy to see that for $\beta > 0$, $\ell \in [0, 0.5)$. Increasing ℓ (or decreasing β) results in more general rules, covering more examples. For $\beta \to 0$ we get the gradient descent technique applied to the exponential loss. This means that the gradient descent produces the most general rules in the sense of coverage.

6.2 Regularization

A decision rule has the form of an n-dimensional rectangle, where n is the number of attributes. It can be shown, that the class of n-dimensional rectangles has Vapnik-Chervonenkis (VC) dimension equal to $2n$, and it does not depend on the number of elementary expressions. Theoretical results [30] suggest that an ensemble with a simple base classifier (with low VC dimension) and high prediction confidence (margin) on the data set generalizes well, regardless of the size of the ensemble. Nevertheless, computational experiments show that the performance of rule ensemble can deteriorate as the number of rules grows, especially for the problems with a high level of noise. Similar phenomenon has been observed for other boosting algorithms, in particular for AdaBoost [27, 31, 32]. Therefore, in order to decrease the influence of the noise, the boosting algorithms should be used with regularization.

As pointed out in many places (see, for example, [19]), a regularized classifier can achieve much better results. The form of regularization, which is particularly useful, is the L_1-penalty, also called *lasso*. In the case of rule ensemble, this would lead to the problem consisting in fitting α_m of all possible rules and minimizing an additional term $\sum_m |\alpha_m|$. To approximate a solution of such a regularized problem, ENDER follows a strategy that is called shrinkage [19]. It consists in shrinking a newly generated rule $r_m(\mathbf{x}) = \alpha_m \Phi_m(\mathbf{x})$ towards rules already present in the ensemble:

$$f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \nu \cdot r_m(\mathbf{x}),$$

where $\nu \in (0, 1]$ is a shrinkage parameter that can be regarded as controlling the learning rate. For small ν, we obtain a solution that is close to the regularized one.

Such an approach works even better, when weak learners are uncorrelated. That is why the procedure for finding Φ_m works on a subsample of original data that is a fraction ζ of the set of training examples, drawn without replacement [33]. Such an approach leads to a set of rules that are more diversified and less correlated. Moreover, finding Φ_m on a subsample reduces the computational complexity. Note, however, that a small ν requires a larger M.

Independently on the fact whether Φ_m was found using a subsample or not, we calculate the value of α_m on *all* training examples. This usually decreases $|\alpha_m|$, so it plays also the role of regularization, and avoids overfitting the rule to the training set.

These above three elements (shrinking, sampling, and calculating α_m on the entire training set) used in ENDER constitute a competitive technique to pruning often used in the rule induction algorithms. Our experiments showed that it significantly improves the accuracy of the classifier.

6.3 Regression Problem

Boosting-like formulation of the rule induction algorithm allows to cope with the problem of regression [34]. In this problem, one usually considers the *squared-error loss*:

$$L_{\text{se}}(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2. \tag{21}$$

It is easy to show that for this loss function, the ENDER algorithm computes the rule response according to:

$$\alpha_m = \arg\min_{\alpha} \sum_{i=1}^{N} (y_i - f_{m-1}(\mathbf{x}_i) - \alpha \Phi_m(\mathbf{x}_i))^2 = \frac{\sum_{\Phi(\mathbf{x}_i)=1} (y_i - f_{m-1}(\mathbf{x}_i))}{\sum_{i=1}^{N} \Phi(\mathbf{x}_i)}, \tag{22}$$

which is the average over the residuals $y_i - f_{m-1}(\mathbf{x}_i)$ for examples covered by the rule. The form of $\mathscr{L}_m(\Phi)$ is, in turn, the following:

$$\mathscr{L}_m(\Phi) = -\frac{\left| \sum_{\Phi(\mathbf{x}_i)=1} (y_i - f_{m-1}(\mathbf{x}_i)) \right|}{\sqrt{\sum_{i=1}^{N} \Phi(\mathbf{x}_i)}}. \tag{23}$$

This form results from putting (22) to (5), and solving the latter with respect to Φ.

As we see, the application of boosted decision rules to solving the regression problem is straightforward.

6.4 Building Single Rules

There remains a question: how to generate the condition part of the rule. This question is associated with another one [28]: how close is the generated rule to the best one. In other words, how good are the procedures building condition parts of the rules.

The procedure used in ENDER follows a typical strategy of many other rule learning algorithms. It gives approximate results, as it is based on a greedy approach, but it is very efficient therefore.

Just to mention, the AQ algorithm follows a different rule induction strategy. Its main idea is to generate the so-called *star* for a selected positive example. The star is a set of candidate condition parts. One of these condition parts is selected by using the beam search in the space of the star members.

Some other procedures could be introduced to ENDER in order to generate better rules. These procedures could aim at improving the performance and/or the interpretability of the rule ensemble.

7 Experiment

We compared five rule induction algorithms: Ripper with sequential covering procedure, SLIPPER, LRI, MLRules, and ENDER with constant step minimization. We selected the following parameters for each method:

- Ripper: we used JRipper (Weka [35] implementation of the algorithm) with default parameters.
- SLIPPER: we set the maximum number of iterations to 500, the rest of parameters was default (we kept the internal cross validation, used to choose the optimal number of rules, switched on).
- LRI: according to the experiment in [21], we set the rule length to 5, froze feature selection after 50 rounds, and chose 200 rules per class and 2 disjuncts, since some previous tests showed that those values work well in practice.
- MLRules: according to the experiment in [23], we set $\zeta = 0.5, \nu = 0.1, M = 500$; those parameters have been optimized on an artificial data set.
- ENDER: according to the experiment in [29], we took the algorithm with constant step minimization $\beta = 0.2$ of the exponential loss (CS-Exp), with other parameters set to $\nu = 0.1$, $\zeta = 0.25$, $M = 500$; those parameters have been optimized on an artificial data set.

We used 20 binary classification problems, all taken from the UCI Repository [36]. Each test was performed using 10-fold cross validation (with exactly the same train/test splits for each classifier), and an average 0-1 loss on testing folds was calculated. The results are shown in Table 1.

Table 1. Test errors and ranks (in parenthesis). In the last row, the average rank is computed for each classifier.

DATA SET	MLRULES	ENDER CS-EXP	SLIPPER	LRI	JRIPPER
HABERMAN	26.2 (2.0)	26.2(3.0)	26.8 (4.0)	27.5(5.0)	25.8(1.0)
BREAST-C	25.9 (1.0)	27.2(3.0)	27.9 (4.0)	29.3(5.0)	26.9(2.0)
DIABETES	24.7 (2.0)	24.6(1.0)	25.4 (4.0)	25.4(3.0)	26.2(5.0)
CREDIT-G	24.1 (3.0)	22.8(1.0)	27.7 (4.0)	23.9(2.0)	28.2(5.0)
CREDIT-A	13.3 (3.0)	12.3(2.0)	17.0 (5.0)	12.2(1.0)	13.9(4.0)
IONOSPHERE	6.0 (2.0)	5.7(1.0)	6.5 (3.0)	6.8(4.0)	12.0(5.0)
COLIC	13.9 (1.0)	14.4(2.0)	15.0 (4.0)	16.1(5.0)	14.4(3.0)
HEPATITIS	16.2 (1.0)	18.8(4.0)	16.7 (2.0)	18.0(3.0)	20.2(5.0)
SONAR	12.0 (1.0)	16.4(3.0)	26.4 (4.0)	14.9(2.0)	29.7(5.0)
HEART-STATLOG	16.7 (1.0)	17.4(2.0)	23.3 (5.0)	19.6(3.0)	20.4(4.0)
LIVER-DISORDERS	27.5 (3.0)	24.9(1.0)	30.7 (4.0)	26.6(2.0)	33.0(5.0)
VOTE	3.4 (1.0)	3.4(2.0)	5.0 (5.0)	3.9(3.0)	4.6(4.0)
HEART-C-2	16.5 (2.0)	15.2(1.0)	19.5 (4.0)	18.5(3.0)	20.5(5.0)
HEART-H-2	18.0 (2.0)	17.3(1.0)	20.0 (5.0)	18.3(3.0)	20.0(4.0)
BREAST-W	3.1 (1.0)	3.6(3.0)	4.3 (4.5)	3.3(2.0)	4.3(4.5)
SICK	1.6 (2.0)	1.8(4.0)	1.6 (1.0)	1.8(5.0)	1.7(3.0)
TIC-TAC-TOE	11.3 (4.0)	8.1(3.0)	2.4 (1.0)	12.2(5.0)	2.8(2.0)
SPAMBASE	4.7 (2.0)	4.6(1.0)	5.9 (4.0)	4.9(3.0)	7.2(5.0)
CYLINDER-BANDS	14.4 (1.0)	19.4(3.0)	21.7 (4.0)	16.5(2.0)	31.5(5.0)
KR-VS-KP	1.0 (3.0)	1.0(2.0)	0.6 (1.0)	3.1(5.0)	1.0(4.0)
AVG. RANK	1.9	2.15	3.63	3.3	4.03

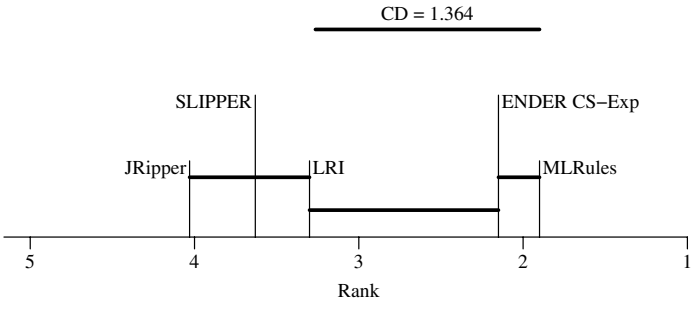

Fig. 2. Critical difference diagram

To compare multiple classifiers on multiple data sets, we follow [37], and apply the Friedman test, which uses ranks of each algorithm to check whether all the algorithms perform equally well (null hypothesis). Friedman statistics gives 27.71, which exceeds the critical value 9.488 (for confidence level 0.05), and thus we can reject the null hypothesis. Next, we proceed to a post-hoc analysis and calculate the *critical difference* (CD) according to the Nemenyi statistics. We obtain CD = 1.364 which means that algorithms with difference in average ranks greater than 1.364 are significantly different. In Figure 2, average ranks were marked on the rank scale, and groups of the classifiers that are not significantly different were connected. This shows that MLRules outperforms LRI, SLIPPER and JRipper. ENDER CS-Exp, in turn, is significantly better than SLIPPER and JRipper. On the other hand, none of the three rule algorithms: LRI, SLIPPER, JRipper, is significantly better than any other. However, the worst results are obtained by JRipper.

The results can be interpreted in the following way. The boosting approach improves, in general, the sequential procedure. The highest improvement is achieved, however, when boosting is combined with regularization, which is applied in both, MLRules and ENDER CS-Exp. As explained above, this is justified by the recent results in theoretical analysis of boosting algorithms. The two other rule ensemble algorithms, SLIPPER and LRI, do not employ such regularization. The former uses rule pruning, but this seems to be not enough.

The main advantage of rules is their simplicity and interpretability, provided that the number of rules in the ensemble is not too high. To ease the interpretation of rules obtained by ENDER, one can apply one of techniques proposed in [22], for example, the post-processing phase in which the rules are refitted by using lasso regularization. Another possibility is to sort the rules in the ensemble using some interestingness measures, as it is done in the case of association rules. Many such measures were already introduced in the literature [38], however, recent results of comparative studies give priority to some Bayesian confirmation measures having desirable properties of symmetry and monotonicity [39, 40].

On the other hand, ENDER can be parameterized in such a way that only few short decision rules are generated – such rules can easily be interpreted while still maintaining good performance. To this aim we have to use no shrinking and no sampling. Moreover, we control the coverage of the rules by using the constant-step minimization.

Table 2. Decision rules for the CREDIT-G data set. The decision part of the rule specifies the class (sign of the rule response), the weight (absolute value of the response) of the rule (in the first parenthesis), and the numbers of training examples that are correctly classified or misclassified by the rule (in the second parenthesis, separated by ":").

#	RULE
1.	*if* CHECKING STATUS $=$ *no checking account* *and* OTHER PAYMENT PLANS $\neq$ *bank* *and* AGE $\in [22.5, 66.5]$ *and* CREDIT AMOUNT ≤ 10924.5, *then* CUSTOMER $=$ *good* (0.853) (302:23)
2.	*if* DURATION ≥ 15.5 *and* PURPOSE $\notin \{$*used car, retraining*$\}$ *and* SAVINGS STATUS ≤ 500 *and* AGE ≤ 63.5 *and* CHECKING STATUS ≤ 200, *then* CUSTOMER $=$ *bad* (0.606) (161:164)
3.	*if* CREDIT HISTORY $=$ *critical/other existing credit* *and* CREDIT AMOUNT ≤ 7806.5 *and* DURATION ≤ 37.5 *and* PURPOSE $\neq$ *education* *and* AGE ≥ 23.5 *and* OTHER PAYMENT PLANS $=$ *none* *and* OTHER PARTIES $\in \{$*none, guarantor*$\}$, *then* CUSTOMER $=$ *good* (0.883) (182:14)
4.	*if* CHECKING STATUS < 0 *and* OTHER PARTIES $=$ *none* *and* SAVINGS STATUS ≤ 500 *and* CREDIT AMOUNT ≥ 438.0 *and* FOREIGN WORKER $=$ *no* *and* JOB $\neq$ *highqualif, selfemp, mgmt* *and* INSTALLMENT COMMITMENT ≥ 1.5 *and* PURPOSE $\neq$ *business* *and* AGE ≤ 66.0, *then* CUSTOMER $=$ *bad* (0.636) (95:48)
5.	*if* PURPOSE $=$ *radio, tv* *and* PROPERTY MAGNITUDE $\neq$ *no known property* *and* AGE $\in [22.5, 65.5]$ *and* CREDIT AMOUNT ≤ 4094.5 *and* CREDIT HISTORY $\neq$ *no credits, all paid* *and* DURATION ≥ 39.0 *and* OTHER PARTIES $\in \{$*none, guarantor*$\}$, *then* CUSTOMER $=$ *good* (0.646) (168:24)

Consider, for example, the CREDIT-G data set. We use ENDER with the exponential loss and constant-step minimization. We limit the number of rules to 5. We do not perform regularization, i.e. no shrinking and no sampling is used. To obtain good performance and interpretable rules, we tune parameter β only.

The rules are presented in Table 2. Parameter β has been set to 0.2. In 10-fold cross-validation, the algorithm obtained the misclassification error equal to 26.2, which is comparable to results obtained by other algorithms (see Table 1). Let us remark that in this data set, 700 examples belong to the class of *good* customers, and 300 examples to the class of *bad* customers. That is why the rules voting for the latter class can cover more examples from the former class (see, the second rule).

8 Conclusions

We have shown how the algorithms for rule induction evolved from the sequential covering approach to the boosted rule ensembles. This latter framework for rule induction algorithms have several important advantages.

Our results confirm the observation given in [20], that boosted rule ensembles are in fact simpler and better-understood formally than other state-of-the-art rule learners. They can be used with different loss functions. The purity measure is defined in a natural way by performing different minimization techniques. One of them allows to control the rule coverage. The proper regularization increases significantly the performance of the algorithm. It is also easy to use the boosted decision rules to regression problems.

There are still, however, several problems waiting for a better solution. One of them is the procedure for building the condition part of a single rule. Another one is the improvement of interpretability of a rule ensemble.

Acknowledgements. The authors wish to acknowledge financial support from the Ministry of Science and Higher Education (grant no. N N519 314435).

References

1. Michalski, R.S.: A Theory and Methodology of Inductive Learning. In: Michalski, R.S., Carbonell, J.G., Mitchell, T.M. (eds.) Machine Learning: An Artificial Intelligence Approach, pp. 83–129. Tioga Publishing, Palo Alto (1983)
2. Clark, P., Niblett, T.: The CN2 induction algorithm. Machine Learning 3, 261–283 (1989)
3. Cohen, W.W.: Fast Effective Rule Induction. In: Proc. of International Conference of Machine Learning, pp. 115–123 (1995)
4. Fürnkranz, J.: Separate-and-Conquer Rule Learning. Artificial Intelligence Review 13(1), 3–54 (1996)
5. Jovanoski, V., Lavrac, N.: Classification Rule Learning with APRIORI-C. In: Proc. of the 10th Portuguese Conference on Progress in Artificial Intelligence, Knowledge Extraction, Multi-agent Systems, Logic Programming and Constraint Solving, London, UK, pp. 44–51. Springer, Heidelberg (2001)
6. Stefanowski, J., Vanderpooten, D.: Induction of Decision Rules in Classification and Discovery-oriented Perspectives. International Journal on Intelligent Systems 16(1), 13–27 (2001)

7. Bazan, J.G.: Discovery of Decision Rules by Matching New Objects Against Data Tables. In: Polkowski, L., Skowron, A. (eds.) RSCTC 1998. LNCS (LNAI), vol. 1424, pp. 521–528. Springer, Heidelberg (1998)
8. Góra, G., Wojna, A.: RIONA: A New Classification System Combining Rule Induction and Instance-based Learning. Fundamenta Informaticae 54, 369–390 (2002)
9. Domingos, P.: Unifying Instance-based and Rule-based Induction. Machine Learning 24, 141–168 (1996)
10. Góra, G., Wojna, A.: Local Attribute Value Grouping for Lazy Rule Induction. In: Alpigini, J.J., Peters, J.F., Skowron, A., Zhong, N. (eds.) RSCTC 2002. LNCS (LNAI), vol. 2475, pp. 405–412. Springer, Heidelberg (2002)
11. Boros, E., Hammer, P.L., Ibaraki, T., Kogan, A., Mayoraz, E., Muchnik, I.: An Implementation of Logical Analysis of Data. IEEE Transactions on Knowledge and Data Engineering 12, 292–306 (2000)
12. Pawlak, Z.: Rough Sets. Theoretical Aspects of Reasoning about Data. Kluwer Academic Publishers, Dordrecht (1991)
13. Słowiński, R. (ed.): Intelligent Decision Support. In: Handbook of Applications and Advances of the Rough Set Theory. Kluwer Academic Publishers, Dordrecht (1992)
14. Grzymala-Busse, J.W.: LERS — A System for Learning from Examples based on Rough Sets. In: Słowiński, R. (ed.) Intelligent Decision Support, Handbook of Applications and Advances of the Rough Sets Theory, pp. 3–18. Kluwer Academic Publishers, Dordrecht (1992)
15. Skowron, A.: Extracting Laws from Decision Tables - A Rough Set Approach. Computational Intelligence 11, 371–388 (1995)
16. Stefanowski, J.: On Rough Set based Approach to Induction of Decision Rules. In: Skowron, A., Polkowski, L. (eds.) Rough Set in Knowledge Discovering, pp. 500–529. Physica Verlag, Heidelberg (1998)
17. Greco, S., Matarazzo, B., Słowiński, R., Stefanowski, J.: An algorithm for induction of decision rules consistent with the dominance principle. In: Ziarko, W.P., Yao, Y. (eds.) RSCTC 2000. LNCS (LNAI), vol. 2005, pp. 304–313. Springer, Heidelberg (2001)
18. Freund, Y., Schapire, R.E.: A Decision-theoretic Generalization of On-line Learning and an Application to Boosting. Journal of Computer and System Sciences 55(1), 119–139 (1997)
19. Hastie, T., Tibshirani, R., Friedman, J.H.: Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer, Heidelberg (2003)
20. Cohen, W.W., Singer, Y.: A Simple, Fast, and Effective Rule Learner. In: Proc. of National Conference on Artificial Intelligence, pp. 335–342 (1999)
21. Weiss, S.M., Indurkhya, N.: Lightweight Rule Induction. In: Proc. of International Conference on Machine Learning, pp. 1135–1142 (2000)
22. Friedman, J.H., Popescu, B.E.: Predictive Learning via Rule Ensembles. Annals of Applied Statistics 2(3), 916–954 (2008)
23. Dembczyński, K., Kotłowski, W., Słowiński, R.: Maximum Likelihood Rule Ensembles. In: Proc. of International Conference on Machine Learning, pp. 224–231 (2008)
24. Błaszczyński, J., Dembczyński, K., Kotłowski, W., Słowiński, R., Szelag, M.: Ensembles of Decision Rules. Foundations of Computing and Decision Sciences 31(3-4), 221–232 (2006)
25. Friedman, J.H.: Greedy Function Approximation: A Gradient Boosting Machine. Annals of Statistics 29(5), 1189–1232 (2001)
26. Breiman, L.: Bagging Predictors. Machine Learning 24(2), 123–140 (1996)
27. Mason, L., Baxter, J., Bartlett, P., Frean, M.: Functional Gradient Techniques for Combining Hypotheses. In: Bartlett, P., Schölkopf, B., Schuurmans, D., Smola, A.J. (eds.) Advances in Large Margin Classifiers, pp. 33–58. MIT Press, Cambridge (1999)
28. Fürnkranz, J.: Rule-based Classification. In: From Local Patterns to Global Models ECML/PKDD 2008 Workshop (2008)

29. Dembczyński, K., Kotłowski, W., Słowiński, R.: A General Framework for Learning an Ensemble of Decision Rules. In: Fürnkranz, J., Knobbe, A. (eds.) From Local Patterns to Global Models ECML/PKDD 2008 Workshop (2008)
30. Schapire, R.E., Freund, Y., Bartlett, P., Lee, W.S.: Boosting the Margin: A New Explanation for the Effectiveness of Voting Methods. Annals of Statistics 26(5), 1651–1686 (1998)
31. Friedman, J.H., Hastie, T., Tibshirani, R.: Additive Logistic Regression: A Statistical View of Boosting. Annals of Statistics (with discussion) 28(2), 337–407 (2000)
32. Dietterich, T.G.: An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization. Machine Learning 40(2), 139–158 (2000)
33. Friedman, J.H., Popescu, B.E.: Importance Sampled Learning Ensembles. Research report, Dept. of Statistics, Stanford University (2003)
34. Dembczyński, K., Kotłowski, W., Słowiński, R.: Solving Regression by Learning an Ensemble of Decision Rules. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2008. LNCS (LNAI), vol. 5097, pp. 533–544. Springer, Heidelberg (2008)
35. Witten, I.H., Frank, E.: Data Mining: Practical machine learning tools and techniques, 2nd edn. Morgan Kaufmann, San Francisco (2005)
36. Asuncion, A., Newman, D.J.: UCI Machine Learning Repository (2007)
37. Demšar, J.: Statistical Comparisons of Classifiers over Multiple Data Sets. Journal of Machine Learning Research 7, 1–30 (2006)
38. Hilderman, R.J., Hamilton, H.J.: Knowledge Discovery and Measures of Interest. Kluwer Academic Publishers, Boston (2001)
39. Greco, S., Pawlak, Z., Słowiński, R.: Can Bayesian confirmation measures be useful for rough set decision rules? Engineering Applications of Artificial Intelligence 17, 345–361 (2004)
40. Brzezińska, I., Greco, S., Słowiński, R.: Mining Pareto-optimal Rules with Respect to Support and Anti-support. Engineering Applications of Artificial Intelligence 20, 587–600 (2007)