A General Framework for Learning an Ensemble of Decision Rules

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

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

Abstract. In the context of binary classification, we analyze a decision rule classifier based on boosting. We consider different loss functions and minimization techniques along with shrinking and sampling. The results show that minimization techniques influence rule construction, particularly rule coverage. The performance of the classifier seems not to be highly affected by the chosen loss function, and the crucial element is proper selection of shrinking and sampling method. The experiment shows the algorithm presented here is competitive to well-known decision rule learners such as SLIPPER, LRI and RuleFit.

1 Introduction

We consider a learning algorithm based on *decision rules* for solving binary classification problems. Such decision rules are simple and interpretable logical patterns of the form: "if *condition* then *decision*". They can be treated as simple classifiers that give a constant response to examples satisfying the condition part, and abstain from the response for other examples.

The problem of induction of decision rules has been widely considered in machine learning. The most popular algorithms were based on sequential covering [1–3]. Recently, another approach to rule induction receives increasing attention. In this approach rules are built using boosting [4] or forward stagewise additive modeling [5]. Such an approach can be seen as a 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 been added. Each rule is fitted by concentrating on examples which were hardest to classify correctly by the rules already present in the ensemble. This is accomplished in terms of minimization of the loss function on the training set. There are several loss functions commonly used in boosting, the most popular being exponential and logit loss functions. A typical procedure for building a single rule resembles generation of decision trees, but only one path from the root to the leaf is considered. The growth of the rule is controlled by a measure that depends on the chosen loss function. In case of exponential loss, one can give an exact formula of the loss minimizer at a given iteration, which determines the measure to be used. The same applies to a formula determining the

response (decision) of the rule. For other loss functions like logit loss, a specific minimization technique has to be applied. The most popular minimization techniques are gradient descent [6] and gradient boosting [7] relying on fitting the negative gradient of the loss function on the training set.

In this paper, we introduce an algorithm for learning an ensemble of decision rules (referred to as ENDER) that can be used with different loss functions and minimization techniques. It consistently uses the same measure (value of the empirical risk) at all stages of the learning procedure: setting the best conditions, stopping the rule's growth and determining the response (decision) of the rule; no additional measures or tools (e.g., impurity measures, pruning procedures) are employed. We also take into account other issues, such as shrinking base learners towards priors and sampling from the whole training set when a single base classifier is built. We thoroughly consider how the learning process and the prediction accuracy is influenced by the characteristic of the loss function, minimization technique, the number of rules, resampling and amount of shrinkage. Up to our knowledge, such analysis has not been yet conducted in the context of decision rules before, however, several rule ensemble learning algorithms exist: SLIPPER [8], LRI [9], RuleFit [10] and MLRules [11]. All these algorithms are quite similar and fall into our framework. The main difference is in the chosen loss function and minimization technique.

1.1 Main Contribution

We verified empirically a performance of ensemble of decision rules with three different loss functions: exponential, logit and sigmoid. We show that the choice of the loss function has only a little impact on the performance.

From our theoretical analysis, it follows that the minimization technique influences the coverage of the rule (number of covered examples). We consider four different minimization techniques, where one of them, *constant-step minimization*, is particularly tailored for decision rules. It relies on building a rule for a fixed constant value of rule response, the step length in the optimization process. We show that the coverage of the rule can be controlled by the step length. When the length approaches zero, this technique is equivalent to gradient descent [6]. We also show the relation between gradient descent and gradient boosting [7] in the context of decision rules. It follows from the analysis that gradient descent produces the most general rules (i.e., rules with the highest coverage). Our theoretical results are also confirmed by the experiment. There is, however, no clear picture which minimization technique gives the best performance, but it seems that gradient boosting is outperformed by the other methods.

We also show experimentally that the main impact on improving the performance is the use of a proper shrinkage and sampling technique. This can be regarded as a type of regularization. This result is rather common for boosting algorithms, however, it is interesting, what parameters of shrinkage and sampling are the best in the case of rules. Moreover, in the introduced algorithm, we compute a response of the rule on *all* training examples, independently on the fact whether a rule was built using a subsample or not. This usually decreases

the response, so it plays also the role of regularization, and avoids overfitting the rule to the training set. These three elements (shrinking, sampling, and calculating response of the rule on the whole training set) constitute an alternative to pruning used very often in induction of decision rules.

1.2 Contents

The paper is organized as follows. In Section 2 we formulate the binary classification problem. Section 3 outlines the introduced algorithm. The theoretical analysis with respect to different loss functions and minimization techniques is given in Sections 4 and 5. Section 6 discusses the connections to other rule induction algorithms. Section 7 contains results of large experiment on artificial data and comparison to the other methods. Section 8 concludes the paper.

2 Problem Statement

We consider a binary classification problem in which the aim is to predict an unknown value of an attribute $y \in \{-1, 1\}$ of an object using known joint values of other attributes $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. Objects for which $y = 1$ are often called "positive examples", where the others are called "negative examples". The task is to find a function $f(\mathbf{x})$ that predicts accurately the value of y. The accuracy of a single prediction is measured by the *loss function* $L(y, f(\mathbf{x}))$ which is 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 (*Bayes risk*) over the joint distribution $P(y, \mathbf{x})$:

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

Therefore, the optimal (risk-minimizing) decision function (or Bayes optimal decision) 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. In the following, we use the linear combination of decision rules.

In binary classification, loss functions can be defined in terms of the margin. Let us assume that $f(\mathbf{x}) \in \mathbb{R}$. Then, the margin is defined as $yf(\mathbf{x})$. Since $y \in \{-1, 1\}$, positive margin means correct classification, and negative – misclassification. The most popular loss is then the so-called margin 0-1 loss:

$$L_{0-1}(yf(\mathbf{x})) = I(yf(\mathbf{x}) < 0), \tag{1}$$

where $I(a)$ is an indicator function (i.e., I(a) = 1 if a is true, otherwise $I(a) = 0$). The expected value of this loss function is simply a misclassification error of $f(\mathbf{x})$ defined as $P(yf(\mathbf{x}) < 0)$. That is why Bayes optimal decision has the following form:

$$f^*(\mathbf{x}) = \arg\min_{f(\mathbf{x})} E_{y|\mathbf{x}} L_{0-1}(yf(\mathbf{x})) = \operatorname{sgn}(2\Pr(y = 1|\mathbf{x}) - 1). \tag{2}$$

It follows that by minimizing the margin 0-1 loss function, one estimates a region in the attribute space, in which the positive class is observed with the higher probability.

There is a problem with minimization of function (1), since it is neither convex, nor differentiable. Moreover, it would be desired that magnitude of $yf(\mathbf{x})$ denoted the credibility of assigning an object to a given class. Therefore, instead of margin 0-1 loss, convex surrogates (upper-bounding the 0-1 loss) are commonly used, such as the exponential and the logit loss, which makes the minimization process easier to cope with. Exponential loss is defined as:

$$L_{\exp}(yf(\mathbf{x})) = \exp(-yf(\mathbf{x})). \tag{3}$$

This loss function is (implicitly) used in AdaBoost [4]. This fact was firstly discovered by [12]. Logit loss:

$$L_{\log}(yf(\mathbf{x})) = \log(1 + \exp(-2yf(\mathbf{x}))) \tag{4}$$

has been applied in LogitBoost [12] and also in Gradient Boosting [7]. These two loss functions have the same Bayes optimal decision:

$$f^*(\mathbf{x}) = \frac{1}{2} \log \frac{\Pr(y = 1|\mathbf{x})}{\Pr(y = -1|\mathbf{x})}, \tag{5}$$

which is the logit transform of the conditional probabilities. Therefore, minimization of these loss functions on the training set can be seen as estimation of conditional probabilities $\Pr(y = 1|\mathbf{x})$. Sign of $f(\mathbf{x})$ estimates in turn the class with higher probability.

Another possibility is to use the sigmoid loss that is a continuous approximation of the 0-1 loss:

$$L_{\text{sigm}}(yf(\mathbf{x})) = \frac{1}{1 + \exp(yf(\mathbf{x}))}. \tag{6}$$

Although not convex, it is differentiable. Bayes optimal decision for (6) is rather aberrant:

$$f^*(\mathbf{x}) = \begin{cases} +\infty & \Pr(y = 1|\mathbf{x}) > \frac{1}{2}, \\ -\infty & \Pr(y = -1|\mathbf{x}) < \frac{1}{2}, \\ \text{arbitrary} & \text{otherwise.} \end{cases}$$

There are theoretical justifications for using this loss function. It is shown in [6] that the upper bound of the misclassification error for such a loss function is tighter than the bound obtained by [13]. Moreover, contrary to the exponential and the logit loss functions, this loss function is bounded within the range $(0, 1)$, and is therefore less sensitive to outliers.

3 Learning an Ensemble of Decision Rules

In this section, we describe the general scheme of learning an ensemble of decision rules. We start with the definition of a single rule, define the ensemble and outline the learning procedure.

Let X_j be a value set of attribute $j \in \{1, \ldots, n\}$ (i.e., the set of all possible values of attribute j). Condition part of the rule consists of a conjunction of elementary expressions of the general form $x_j \in S_j$, where x_j is the value of object $\mathbf{x}$ on attribute j and S_j is a subset of X_j. In particular, we assume that elementary expressions are in 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 value set of j-th attribute. Let Φ be the set of elementary expressions constituting the condition part of the rule, and $\Phi(\mathbf{x})$ be a function indicating whether an object $\mathbf{x}$ satisfies the condition part Φ, i.e. $\Phi(\mathbf{x}) = I(\mathbf{x} \in \Phi)$. In other words, $\Phi(\mathbf{x})$ defines an arbitrary axis-parallel region in the attribute space. We say that a rule *covers* an object if $\Phi(\mathbf{x}) = 1$. The decision (response), denoted by α, is a real non-zero 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. We assume that the classification function is a linear combination of M decision rules:

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

where α_0 is a constant value, which can be interpreted as a default rule, covering the whole attribute space. Classification is made simply by computing $\operatorname{sgn}(f(\mathbf{x}))$.

The construction of an optimal combination of rules minimizing the empirical risk is a hard optimization problem. We therefore follow forward stagewise additive modeling that results in an iterative procedure in which rules are added one by one. We start with the default rule defined as:

$$\alpha_0 = \arg\min_{\alpha} \sum_{i=1}^{N} L(y_i, \alpha). \tag{7}$$

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:

$$r_m(\mathbf{x}) = \arg\min_{\Phi,\alpha} \sum_{i=1}^{N} L(y_i(f_{m-1}(\mathbf{x}_i) + \alpha\Phi(\mathbf{x}_i))). \tag{8}$$

Since the exact solution of (8) is still computationally hard, we proceed in two steps.

Step 1. Find Φ_m by minimizing a functional $\mathcal{L}_m(\Phi)$ in a greedy manner:

$$\Phi_m = \arg\min_{\Phi} \mathcal{L}_m(\Phi). \tag{9}$$

The particular form of this functional is derived from (8) and depends on the chosen loss function and minimization technique (see Section 5). The greedy procedure used for finding Φ_m resembles the way the decision trees are generated, but we look for only one path from the root to the leaf. At the beginning, Φ_m is empty and in each subsequent step an elementary expression $x_j \in S_j$ is added to Φ_m until $\mathcal{L}_m(\Phi_m)$ cannot be decreased. Let us underline that, contrary to the generation of decision trees, a minimal value of $\mathcal{L}_m(\Phi)$ is a natural stop criterion.

Step 2. Find α_m, the solution to the following line-search problem:

$$\alpha_m = \arg\min_{\alpha} \sum_{i=1}^{N} L(y_i(f_{m-1}(\mathbf{x}_i) + \alpha\Phi_m(\mathbf{x}_i)). \tag{10}$$

In several cases the computation of α_m is straightforward, because analytical solution for (10) can be given, otherwise, an approximate solution has to be found (see Section 4).

Regularization. As pointed out in many places (see, for example, [5]), 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 empirical risk minimization taking into account all possible rules with additional term $\sum_m |\alpha_m|$. To approximate a solution of such regularized problem, one can follow a strategy that is called shrinkage [5]. 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, one can 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 of η training examples, drawn without replacement [14]. 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 small ν requires larger M. Independently on the fact whether Φ_m was found using a subsample or not, value of α_m is calculated on *all* training examples in the introduced algorithm. This usually decreases $|\alpha_m|$, so it plays also the role of regularization, and avoids overfitting the rule to the training set. These three elements (shrinking, sampling, and calculating α_m on the whole training set) constitute a competitive technique to pruning. Our experiments showed that it significantly improves the accuracy of the classifier.

Algorithm 1 (referred to as ENDER) outlines the whole procedure for constructing an ensemble of decision rules.

Algorithm 1: Ensemble of decision rules – ENDER

input : set of training examples $\{y_i, \mathbf{x}_i\}_1^N$,
$L(y\mathbf{x})$ – loss function,
M – number of decision rules to be generated,
$\mathcal{L}_m(\Phi)$ – minimization technique that works on
a fraction η of training examples, drawn without replacement,
ν – shrinkage parameter.
output: default rule α_0, 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**
  $\Phi_m = \arg\min_\Phi \mathcal{L}_m(\Phi)$
  $\alpha_m = \arg\min_\alpha \sum_{i=1}^N L(y_i(f_{m-1}(\mathbf{x}_i) + \alpha\Phi_m(\mathbf{x}_i)))$
  $r_m(\mathbf{x}) = \alpha_m\Phi_m(\mathbf{x})$
  $f_m(\mathbf{x}) = f_{m-1} + \nu \cdot r_m(\mathbf{x})$
end

4 Rule Response Calculation

In this section, we review methods of solving the line search problem (10) to determine the decision α_m of the rule r_m, when the condition part Φ_m of the rule is already known. Depending on the loss function used, one can obtain analytical solution or do simple numerical optimization.

For the exponential loss function, the solution of (10) is given by:

$$\alpha_m = \arg\min_\alpha \sum_{i=1}^N e^{-y_i(f_{m-1}(\mathbf{x}_i)+\alpha\Phi_m(\mathbf{x}_i))} = \frac{1}{2}\log\frac{\sum_{y_i=1}\Phi(\mathbf{x}_i)e^{-f_{m-1}(\mathbf{x}_i)}}{\sum_{y_i=-1}\Phi(\mathbf{x}_i)e^{f_{m-1}(\mathbf{x}_i)}}. \quad (11)$$

In case of the logit loss there is no analytical solution to (10). To speed up the computations, instead of numerically solving the line search problem, a single Newton-Raphson step is performed, similarly as in [7]:

$$\alpha_m = -\frac{\sum_{\Phi(\mathbf{x})=1}\frac{\partial}{\partial\alpha}L_{\log}(y_i(f_{m-1}(\mathbf{x}) + \alpha\Phi_m(\mathbf{x})))}{\sum_{\Phi(\mathbf{x})=1}\frac{\partial^2}{\partial\alpha^2}L_{\log}(y_i(f_{m-1}(\mathbf{x}) + \alpha\Phi_m(\mathbf{x})))}\Bigg|_{\alpha=0}. \quad (12)$$

Sigmoid loss is another loss function for which no analytical solution to (10) exists. Moreover, due to non-convexity of this function one should avoid the Newton-Raphson method. Nevertheless, to speed up the computations, we do not solve the line search problem, but instead we perform one step of a small constant length γ along the direction of the negative gradient.

5 Minimization Techniques

In this section, we derive from (8) the form of the functional $\mathcal{L}_m(\Phi)$ minimized in (9). There have been different minimization techniques considered for this

task within boosting [6, 7], some of which will be reviewed here in the context of decision rules. As we show, the minimization technique highly influences the coverage of a rule (i.e., number of covered examples). In particular, we introduce a technique that relies on building a rule for a fixed constant value of rule response. This constant value is a step in the minimization procedure. We prove that the coverage of the rule can be controlled by the step length.

Simultaneous minimization. Simultaneous minimization [15] of Φ and α in formula (8) is possible only for the exponential loss, since only for this function the exact solution to (10) is known. One puts the optimal value of α_m given by (11) into (8), and obtains expression $\mathcal{L}_m(\Phi)$ depending only on Φ:

$$\mathcal{L}_m(\Phi) = 2 \times \sqrt{\left(\sum_{y_i=1} \Phi(\mathbf{x}_i) w_i^{(m)}\right)\left(\sum_{y_i=-1} \Phi(\mathbf{x}_i) w_i^{(m)}\right)} + \sum_{\Phi(\mathbf{x}_i)=0} w_i^{(m)}, \quad (13)$$

where $w_i^{(m)} = e^{-y_i f_{m-1}(\mathbf{x}_i)}$ can be treated as the weight of the i-th training example in the m-th iteration.

Gradient Descent. This technique, in contrary to the simultaneous minimization, can be used with any differentiable loss function [6]. It approximates (8) up to the first order with respect to α:

$$r_m(\mathbf{x}) \simeq \arg\min_{\Phi,\alpha} \sum_{i=1}^{N} \left(L(y_i f_{m-1}(\mathbf{x}_i)) + y_i \alpha \Phi(\mathbf{x}_i) w_i^{(m)} \right), \quad (14)$$

where

$$w_i^{(m)} = -\frac{\partial L(y_i f_{m-1}(\mathbf{x}_i))}{\partial (y_i f_{m-1}(\mathbf{x}_i))}.$$

It is easy to see that the optimal solution with respect to Φ is obtained by minimizing:

$$\mathcal{L}_m(\Phi) = -\sum_{\Phi(\mathbf{x}_i)=1} y_i \alpha w_i^{(m)}, \quad (15)$$

since $\sum_{i=1}^{N} L(y_i f_{m-1}(\mathbf{x}_i))$ is constant at a given iteration, thus does not change the solution. Let

$$R_+ = \{i\colon\ \Phi(\mathbf{x}_i) = 1 \wedge \alpha y_i > 0\}$$

denote a set of examples "correctly classified" by the rule (for $y_i = 1$, α should be > 0, and for $y_i = -1$, α should be < 0). Analogously,

$$R_- = \{i\colon\ \Phi(\mathbf{x}_i) = 1 \wedge \alpha y_i < 0\}$$

denotes a set of examples "misclassified" by the rule. Then, minimization of (15) is equivalent to minimization of:

$$-\sum_{i \in R_+} w_i^{(m)} + \sum_{i \in R_-} w_i^{(m)}, \quad (16)$$

because magnitude $|\alpha|$ does not change a solution. Then, in the case of learning with decision rules, one can prove the following:

Theorem 1. *Minimization of (14) is equivalent to minimization of:*

$$\mathcal{L}(\Phi) = \sum_{i \in R_-} w_i^{(m)} + \frac{1}{2} \sum_{\Phi(\mathbf{x}_i)=0} w_i^{(m)}, \tag{17}$$

Proof. Let us remark that

$$\sum_{i \in R_+} w_i^{(m)} = \sum_{i=1}^{N} w_i^{(m)} - \sum_{i \in R_-} w_i^{(m)} - \sum_{\Phi(\mathbf{x}_i)=0} w_i^{(m)}. \tag{18}$$

Since $\sum_{i=1}^{N} w_i^{(m)}$ is constant at a given iteration, it can be added or subtracted from (16) without any change in the optimization process. Thus, we finally obtain that a subject to minimize is:

$$\sum_{i \in R_-} w_i^{(m)} + \frac{1}{2} \sum_{\Phi(\mathbf{x}_i)=0} w_i^{(m)},$$

as claimed. □

This theorem has a nice interpretation: the first term of (17) corresponds to examples "misclassified" by the rule, while the second term – to examples which are not classified by the rule at all. Value $\frac{1}{2}$ plays the role of a penalty for abstaining from classification and establishes a trade-off between not classified and misclassified examples.

Let us also remark that minimization of (16) could be reformulated to the minimization of the following term:

$$-\left| \sum_{\Phi(\mathbf{x}_i)=1} y_i w_i^{(m)} \right|, \tag{19}$$

because a sing of α may be established afterwards. This formulation will serve us the connection with gradient boosting.

Gradient boosting. This technique [7] was originally introduced as minimization of the squared-error between base classifier response and the negative gradient of the loss function, what can be expressed in the case of rules as:

$$r_m(\mathbf{x}) \simeq \arg\min_{\Phi,\alpha} \left(\sum_{\Phi(\mathbf{x}_i)=1} \left(y_i w_i^{(m)} - \alpha \right)^2 + \sum_{\Phi(\mathbf{x}_i)=0} \left(y_i w_i^{(m)} \right)^2 \right). \tag{20}$$

The following theorem establishes the connection between gradient descent and gradient boosting.

Theorem 2. *Minimization of (20) is equivalent to minimization of:*

$$\mathcal{L}_m(\Phi) = -\frac{\left|\sum_{\Phi(\mathbf{x}_i)=1} y_i w_i^{(m)}\right|}{\sqrt{\sum_{i=1}^N \Phi(\mathbf{x}_i)}}, \tag{21}$$

which is equivalent to (19) normalized by the square root of the rule coverage.

Proof. The minimization problem defined in (20) can be solved for:

$$\alpha = \frac{\sum_{\Phi(\mathbf{x}_i)=1} y_i w_i^{(m)}}{\sum_{i=1}^N \Phi(\mathbf{x}_i)}, \tag{22}$$

and by putting (22) into (20), and performing some simple calculations, we obtain:

$$r_m(\mathbf{x}) \simeq \arg\min_{\Phi} \left(\sum_{i=1}^N \left(y_i w_i^{(m)} \right)^2 - \frac{1}{\sum_{i=1}^N \Phi(\mathbf{x}_i)} \left(\sum_{\Phi(\mathbf{x}_i)=1} y_i w_i^{(m)} \right)^2 \right).$$

After removing the first term under $\arg\min$, which is constant, and taking the square root of the second term (which does not affect the minimization) we get:

$$-\frac{\left|\sum_{\Phi(\mathbf{x}_i)=1} y_i w_i^{(m)}\right|}{\sqrt{\sum_{i=1}^N \Phi(\mathbf{x}_i)}},$$

as claimed.

Theorem 2 suggests that gradient boosting results in more specific rules than gradient descent, covering smaller regions in the attribute space.

Constant-step minimization. Finally, we present a novel technique, particularly tailored for decision rules, that consists in minimization of the loss function with the constant step. In other words, we restrict α in (8) to $\alpha \in \{-\beta, \beta\}$, where β is a fixed parameter of the algorithm. Then, (8) becomes:

$$r_m(\mathbf{x}) = \arg\min_{\Phi, \pm\beta} \left(\sum_{\Phi(\mathbf{x}_i)=1} L(y_i(f_{m-1}(\mathbf{x}_i) \pm \beta)) + \sum_{\Phi(\mathbf{x}_i)=0} L(y_i f_{m-1}(\mathbf{x}_i)) \right). \tag{23}$$

The above formula can be used with any loss function, since it involves calculating two loss values at points $f_{m-1}(\mathbf{x}_i) \pm \beta$. It is also natural for sigmoid loss (6), for which we approximate (10) by a constant step γ along the direction of the negative gradient.

In the case of the exponential loss, one can prove an interesting theorem which clearly shows, that this approach generalizes gradient descent technique.

Theorem 3. *Solution of (23) for exponential loss (3) and step length β is equivalent to minimization of*

$$\mathcal{L}_m(\Phi) = \sum_{i \in R_-} w_i^{(m)} + \ell \sum_{\Phi(\mathbf{x}_i)=0} w_i^{(m)}, \tag{24}$$

where R_- is defined as before, and:

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

Proof. It follows from putting (3) into (23):

$$r_m(\mathbf{x}) = \arg\min_{\Phi,\pm\beta} \left(\sum_{i \in R_+} w_i^{(m)} e^{-\beta} + \sum_{i \in R_-} w_i^{(m)} e^{\beta} + \sum_{\Phi(\mathbf{x}_i)=0} w_i^{(m)} \right). \tag{25}$$

Since $\sum_{i \in R_+} w_i^{(m)} = \sum_{i=1}^{N} w_i^{(m)} - \sum_{i \in R_-} w_i^{(m)} - \sum_{\Phi(\mathbf{x}_i)=0} w_i^{(m)}$, one can minimize:

$$\mathcal{L}_m(\Phi) = (e^{\beta} - e^{-\beta}) \sum_{i \in R_-} w_i^{(m)} + (1 - e^{-\beta}) \sum_{\Phi(\mathbf{x}_i)=0} w_i^{(m)} + e^{-\beta} \sum_{i=1}^{N} w_i^{(m)}. \tag{26}$$

The last element does not change the solution, so it is sufficient to minimize the first two terms. Moreover, dividing (26) by $(e^{\beta} - e^{-\beta})$ we obtain:

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

where

$$\ell = \frac{1-e^{-\beta}}{e^{\beta}-e^{-\beta}} \qquad \beta = \log\frac{1-\ell}{\ell}.$$

as claimed. □

It is easy to see that for $\beta > 0$, $\ell \in [0, 0.5)$. Expression (24) has a similar interpretation as (17), but with varying value of ℓ. Increasing ℓ (or decreasing β) results in more general rules, covering more examples. For $\beta \to 0$ we get a gradient descent technique applied to exponential loss. This means that gradient descent produces the most general (in the sense of coverage) rules.

For other loss functions, one can prove the following theorem by expanding $L(y(f(\mathbf{x}) \pm \beta))$ with respect to $\pm\beta$ up to the second order.

Theorem 4. *Minimization of (23) for any twice differentiable loss function $L(yf(\mathbf{x}))$ and any β is equivalent to the minimization of:*

$$\mathcal{L}_m(\Phi) = \sum_{i \in R_-} w_i^{(m)} + \frac{1}{2} \sum_{\Phi(\mathbf{x}_i)=0} \left(w_i^{(m)} - \beta v_i^{(m)} \right), \tag{27}$$

where R_- is defined as before, and

$$w_i^{(m)} = -\frac{\partial L(y_i f_{m-1}(\mathbf{x}_i))}{\partial (y_i f_{m-1}(\mathbf{x}_i))}, \quad v_i^{(m)} = \frac{1}{2}\frac{\partial^2 L(y_i f_{m-1}(\mathbf{x}_i) + y_i\gamma)}{\partial (y_i f_{m-1}(\mathbf{x}_i) + y_i\gamma)^2}, \tag{28}$$

for some $\gamma \in [0, \beta]$.

Proof. It follows from Taylor's expansion of $L(y(f(\mathbf{x}) \pm \beta))$ with respect to $\pm\beta$ up to the second order. The formula for $\mathcal{L}_m(\Phi)$ is then:

$$\mathcal{L}_m(\Phi) = \sum_{i \in R_+} \left(L_i + \beta w_i^{(m)} + \beta^2 v_i^{(m)} \right) + \sum_{i \in R_-} \left(L_i - \beta w_i^{(m)} + \beta^2 v_i^{(m)} \right) + \sum_{\Phi(\mathbf{x}_i)=0} L_i,$$

where $L_i = L(y_i f_{m-1}(\mathbf{x}_i))$. After some simple transformations similar to that from the proof of Theorem 3, one proves the thesis. □

As above, β establishes a trade-off between misclassified and unclassified examples. Values $w_i^{(m)}$ are always positive, since the loss function is decreasing. If the loss function is convex (e.g., exponential or logit loss) $v_i^{(m)}$ is also positive, therefore increasing β decreases the penalty for abstaining from classification, which leads to smaller and more specific rules. Notice, that for $\beta \to 0$ expression (27) boils down to the gradient descent technique. The situation changes if the loss function is not convex, which is the case of a sigmoid loss. Sigmoid loss is convex for $yf(\mathbf{x}) > 0$ and concave for $yf(\mathbf{x}) < 0$; therefore, as β increases, uncovered examples satisfying $y_i f_{m-1}(\mathbf{x}_i) > 0$ ("correctly classified") are penalized less, while the penalty for uncovered "misclassified" examples ($y_i f_{m-1}(\mathbf{x}_i) < 0$) increases. This leads to the following conclusion: although the rule covers only a part of the examples, with respect to uncovered examples it still tries to make a small error; remark that the weights of the uncovered examples depend on the curvature of the function (second derivative) rather than on the slope (first derivative).

6 Discussion and Related Works

Let us relate ENDER to existing algorithms, starting with SLIPPER [8]. This is the first boosted rule learner. It was originally introduced as an instance of AdaBoost with confidence-rated predictions [15]. This corresponds to ENDER solving (11) and (13). The difference is that SLIPPER uses pruning when generating a single rule and ENDER uses shrinkage and resampling instead, with rule response calculated over all training examples. The latter approach results in higher accuracy as demonstrated later.

LRI uses a specific reweighting schema (cumulative error), similar to Breiman's Arc-xf algorithm [16]. For the two-class problem, however, this method can also be explained in the context of the loss function minimization, as it was done in [6] for Arc-xf. It follows that LRI minimizes polynomial loss function by 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.

MLRules [11] are derived from the maximum likelihood principle, but can also be seen as minimization of the logit loss (4) by a gradient descent technique. They are also distinguished by elegant generalization to multi-class problems.

The main difference between RuleFit [10] and the above algorithms is that this algorithm does not generate decision rules directly. First, decision trees are used as base classifiers, and then in the second phase of the algorithm, rules are produced from the resulting trees. Finally, a rule ensemble is fitted by gradient directed regularization that aims at selecting the most relevant rules. There is also a possibility to include in this fitting procedure original attributes as basis functions to complement the rule ensemble with a linear part. RuleFit can utilize a variety of loss functions, because it uses gradient boosting technique for fitting trees. Classification problems are originally solved using squared-error ramp loss.

Because RuleFit uses decision trees as base classifiers, this is a right place to discuss the difference between tree-based and rule-based ensembles. The first difference is that there exists a natural stop criterion for rule construction. This is just the minimal value of $\mathcal{L}_m(\Phi)$ that takes into account the trade-off between covered and uncovered training examples. In case of decision trees, one has to define several additional parameters, such as number of terminal nodes, minimal number of training examples in a terminal node, or to perform pruning. The output of the ensemble of decision trees and decision rules is quite similar. For both, this is a linear combination of regions $\Phi(\mathbf{x})$ in the attribute space. The difference is that in the case of rules, each region defined by $\Phi(\mathbf{x})$ is built to be optimal, taking into account all previously generated rules. This is not the case of decision trees, where in each iteration several regions are identified. Moreover, using rules one can generate regions that are hardly obtained by decision trees.

At the end, let us relate boosted rules to sequential covering. This approach relies on learning a rule that covers a part of given training examples, removes the covered examples from the training set and repeats this step until no examples remain. SLIPPER is often referred to as weighted sequential covering, because the examples are not removed totally but their weights decrease (or increase in the case of misclassification). It is also easy to show that sequential covering can be simulated by ENDER. Let us consider a simple heuristic that covers examples from one class only. Moreover, let us use the margin 0-1 loss defined by (1). For such a setting, the value of the 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.

7 Experimental Results

We perform an experiment in two parts. In the first part, we exhaustively test ENDER with different settings on artificial data and draw conclusions about the values of the parameters. In the second part, we compare four variants of ENDER

with existing rule ensemble learning methods: SLIPPER, LRI and RuleFit. The comparison is made on the real datasets taken from the UCI repository [17].

7.1 Artificial Data

Let us briefly describe how the artificial data were generated. To this end, examples $\mathbf{x} \in \mathbb{R}^n$ were drawn according to the normal distribution, $\mathbf{x} \sim N(0, \mathbf{I})$, where $\mathbf{I}$ is a unit matrix of size n. We assume that the "target" function $h(\mathbf{x}) \in \mathbb{R}$, modeling conditional probability distribution $P(y|\mathbf{x})$, satisfies the following:

$$\log \frac{P(y=1|\mathbf{x})}{P(y=-1|\mathbf{x})} = \beta h(\mathbf{x}),$$

where β corresponds to the level of noise, measured by the Bayes risk $R^* = \mathbb{E}[f^*(\mathbf{x})]$ (i.e., there is one-to-one correspondence between β and R^*). In the main experiment we set $R^* = 0.1$. We define the target function $h(\mathbf{x})$ as:

$$h(\mathbf{x}) = x_1 - x_2 + 0.2(x_3 - x_4) + 5e^{-(x_5^2+x_6^2+0.2x_7^2)} - 5\prod_{j=8}^{10} I(-0.5 \le x_j \le 0.5) + I(x_{11} \ge 0 \wedge x_{12} \ge 0) - I(x_{13} \ge 0 \wedge x_{14} \ge 0) + \theta, \quad (29)$$

where threshold θ was chosen so that the prior probabilities of both classes were equal: $P(y=1) = P(y=-1)$. Notice that target function contains linear terms (which are hard to approximate by trees and rules), Gaussian term (ball in the coordinate origin), cube and two rectangles. Later we will also add some irrelevant attributes $x_{15}, x_{16}, \ldots$ which do not affect the target function.

To select the values of parameters, we drew 30 training and testing sets of size $N = 1000$. We tested simultaneous minimization with exponential loss, and three further minimization techniques (constant-step, gradient descent and gradient boosting) with all three loss functions (exponential, sigmoid, logit). For each algorithm we tried all combinations of the following values of the parameters: $\nu \in \{1, 0.5, 0.25, 0.1, 0.05\}$, $\eta \in \{1, 0.75, 0.5, 0.25\}$. Moreover, the constant-step minimization was tested with parameters $\beta \in \{1, 0.5, 0.2, 0.1\}$. We varied M, the size of the ensemble, from 1 to 1000, thus obtaining for each classifier the accuracy on testing set as a function of M. Such functions can be easily shown in the form of "accuracy curves". Using these curves, we were able to choose for each loss function the best minimization technique and the best values of the parameters. The exponential loss was an exception, where the simultaneous minimization was treated separately from the other techniques. Thus we ended up with four algorithms:

- simultaneous minimization with exponential loss (SM-Exp): ν=0.1, η=0.25,
- constant-step with exponential loss (CS-Exp): $\beta = 0.2$, $\nu = 0.1$, $\eta = 0.25$,
- constant-step with logistic loss (CS-Log): $\beta = 0.2$, $\nu = 0.1$, $\eta = 0.25$,
- constant-step with sigmoid loss (CS-Sigm): $\beta = 0.2$, $\nu = 0.2$, $\eta = 0.5$.

Notice that neither gradient descent nor gradient boosting were selected. Gradient descent obtained similar result to constant-step (they are, in fact, very similar), but the gradient boosting was much worse. The curves for each of the best classifiers are shown in top left panel of Figure 1. It follows from the figure that none of the classifiers outperforms the others in a significant way. Sigmoid loss tends to minimize the error slower than other loss functions, on both training and testing set, however for $M = 1000$ it achieves the same accuracy. Notice that CS-Exp does not decrease the training error as rapidly as SM-Exp, yet the characteristics of both algorithms on the testing set are similar. In the top right panel of Figure 1 we show learning curves, the accuracies of the classifiers as functions of the sample size up to $N = 10000$; all classifiers decrease the testing error, but SM-Exp seems to gain the most as N increases.

From the above experiment, it is hard to observe a supremacy of one of the loss functions. In the next step, we investigate how the performance of classifiers depends on shrinkage ν and subsample size η. We focus on SM-Exp, but the relationships are similar for other loss functions and minimization techniques. The bottom panels of Figure 1 show both dependencies. It follows that shrinkage does improve the accuracy, similarly as it was shown in [5]. Nevertheless, too strong shrinkage may lead to a very slow learning rate (see the black curve for $\nu = 0.01$). We found out that the optimal range of shrinkage is 0.1-0.2. It also follows that sampling has a positive impact on the accuracy – even small values of η (≤ 0.5) seem to work very well.

Let us examine the behavior of classifiers if we increase the Bayes risk up to the level of 0.3. This is shown on the top left panel in Figure 2. Obviously, the testing error of the classifiers was increased. However, the shape of accuracy curve for all the classifiers has not changed: we do not observe any significant overfitting. At first sight, it looks contrary to what has usually been observed in boosting experiments: the exponential loss tends to be prone to overfitting since it focuses too much on the incorrectly classified examples, which are typically noise; sigmoid loss should behave best, since it gives up on the hardest examples (because of the shape of this loss function). No such behavior is observed. The top right panel in Figure 2 sheds some light on this phenomenon. The only difference is that the decision of the rule is now calculated on the subsample rather than the whole training set. This leads to severe overfitting of all classifiers except that with the sigmoid loss. Thus, we see that calculating the response on all examples makes the classifier robust to noise: the rules overfitted to the data, will get their responses (decisions) close to 0. This also, at least partially, explains why all loss functions behave roughly the same: since the values of the rule responses are relatively small, the ensemble function $f(\mathbf{x})$ is in the vicinity of 0 for most of the examples and in this range, all loss functions share a similar characteristic (the main difference between the loss functions is for large negative values of the margin).

We also considered adding some irrelevant attributes to the problem. The bottom left panel in Figure 2 shows that the presence of 20 irrelevant attributes

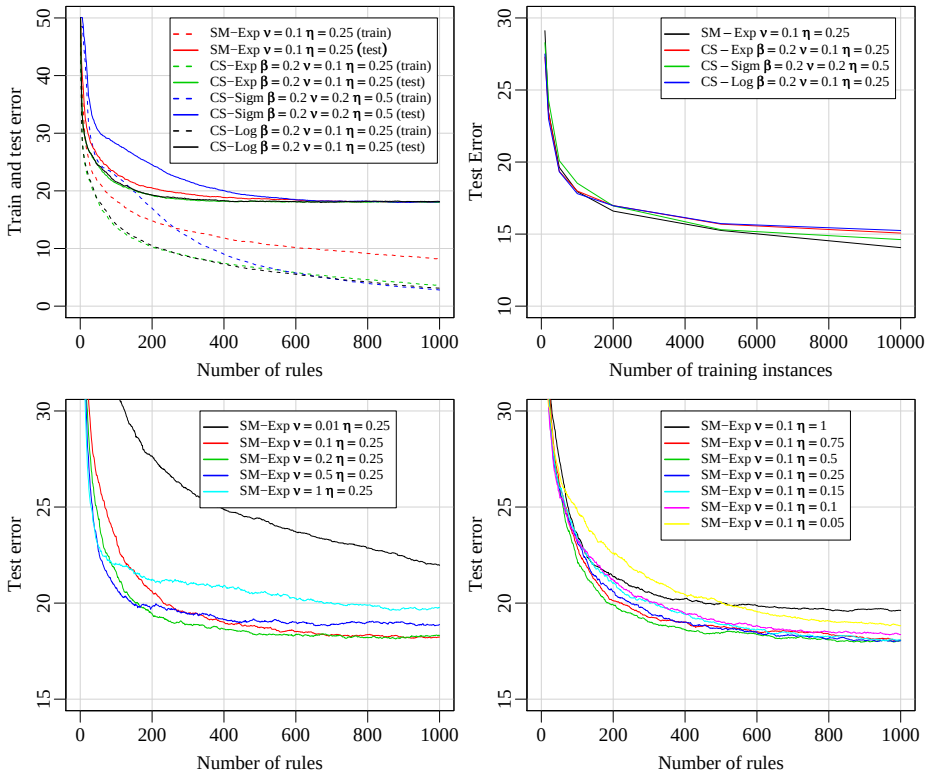

Fig. 1. Experiments on artificial data. Top left: accuracy curves for the best algorithms, top right: learning curves, bottom left: SM-Exp with varying ν and constant $\eta = 0.25$, bottom right: SM-Exp with varying η and constant $\nu = 0.1$.

did not affect ENDER much, regardless of the kind of loss function was used. Notice the similarity between this plot and the top left one in Figure 1.

Finally, the bottom right plot in Figure 2 shows the coverage of the rules for different minimization techniques applied to the exponential loss. We observe what has already been anticipated by Theorem 3, that step length determines the coverage of the rule. We have already noticed that the best prediction accuracy is achieved for the small (but non-zero) step length 0.1-0.2: neither small and well fitted nor very general rules achieve the best prediction performance. Gradient boosting, in turn, produces rules with much lower coverage than gradient descent (see Theorem 2). It is interesting that SM-Exp also generates smaller rules, but achieves better performance. Notice that the possibility of controlling the coverage may be very interesting and helpful in interpreting the meaning of a rule.

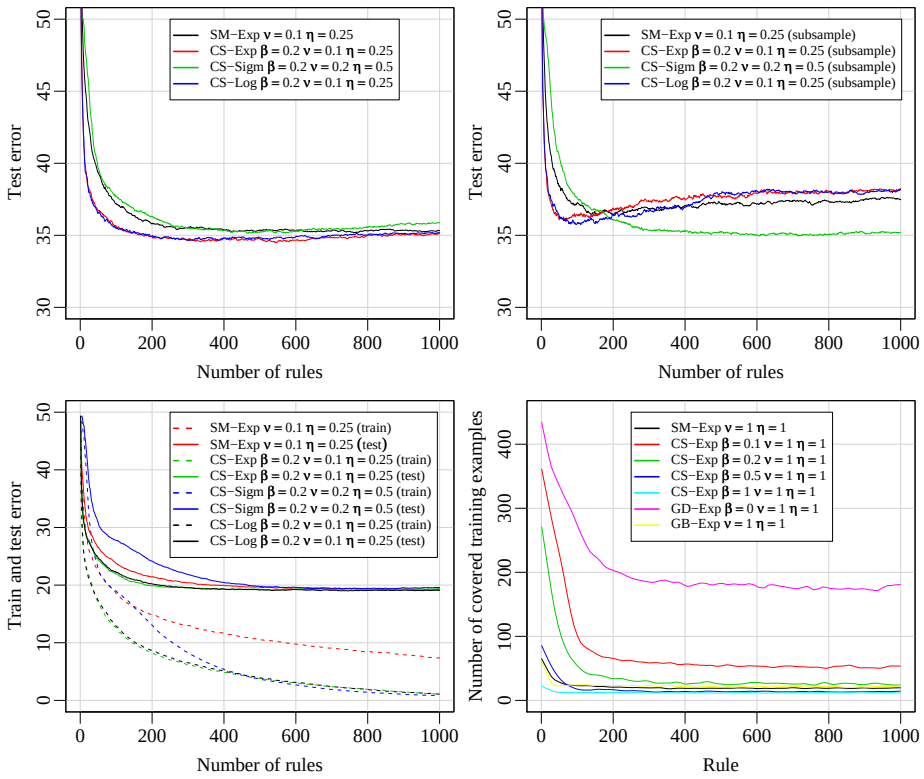

Fig. 2. Experiments on artificial data. Top left: accuracy curves for Bayes risk 0.3 (rule response calculated over the whole training set), top right: rule response calculated on subsample (Bayes risk 0.3), bottom left: accuracy curves for a problem with 20 additional irrelevant features, bottom right: coverage of rules (lines are smoothed).

7.2 Real Data

In the second part of the experiment, we compare ENDER with already existing approaches to learning rule ensembles: LRI, SLIPPER and RuleFit. We selected the following parameters for each method:

- SLIPPER: we set the maximum number of iterations to 500, the rest of parameters remains default (we kept the internal cross validation, used to choose the optimal number of rules, switched on).
- LRI: according to the experiment in [9], we set 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.
- RuleFit: according to the experiment in [10], we chose mixed rule-linear mode, set average tree size to 4, increased number of trees to 500, and chose sample fraction as $\min\{0.5n, 100 + 6\sqrt{n}\}/n$.

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

Dataset	CS-Log	SM-Exp	CS-Exp	CS-Sigm	SLIPPER	LRI	RuleFit
haberman	0.268 (4.5)	0.255(1.0)	0.262(3.0)	0.258(2.0)	0.268 (4.5)	0.275(7.0)	0.272(6.0)
breast-c	0.283 (5.0)	0.279(3.0)	0.272(1.0)	0.273(2.0)	0.279 (4.0)	0.293(6.0)	0.297(7.0)
diabetes	0.245 (2.0)	0.246(3.5)	0.246(3.5)	0.236(1.0)	0.254 (6.0)	0.254(5.0)	0.262(7.0)
credit-g	0.233 (2.0)	0.235(3.0)	0.228(1.0)	0.242(5.0)	0.277 (7.0)	0.239(4.0)	0.259(6.0)
credit-a	0.135 (4.5)	0.135(4.5)	0.123(2.0)	0.138(6.0)	0.17 (7.0)	0.122(1.0)	0.132(3.0)
ionosphere	0.063 (3.0)	0.06 (2.0)	0.057(1.0)	0.065(4.5)	0.065 (4.5)	0.068(6.0)	0.085(7.0)
colic	0.15 (5.0)	0.147(3.5)	0.144(2.0)	0.128(1.0)	0.15 (6.0)	0.161(7.0)	0.147(3.5)
hepatitis	0.195 (7.0)	0.182(4.0)	0.188(5.0)	0.162(1.0)	0.167 (2.0)	0.18 (3.0)	0.194(6.0)
sonar	0.168 (5.0)	0.154(3.0)	0.164(4.0)	0.145(1.0)	0.264 (7.0)	0.149(2.0)	0.197(6.0)
heart-statlog	0.167 (1.0)	0.17 (2.0)	0.174(3.5)	0.174(3.5)	0.233 (7.0)	0.196(6.0)	0.185(5.0)
liver-disorders	0.264 (4.0)	0.258(3.0)	0.249(1.0)	0.249(2.0)	0.307 (7.0)	0.266(5.0)	0.307(6.0)
vote	0.032 (1.0)	0.034(2.5)	0.034(2.5)	0.046(5.0)	0.05 (6.0)	0.039(4.0)	0.05 (7.0)
heart-c-2	0.169 (4.0)	0.155(3.0)	0.152(1.0)	0.155(2.0)	0.195 (7.0)	0.185(5.0)	0.189(6.0)
heart-h-2	0.17 (1.0)	0.176(3.0)	0.173(2.0)	0.193(6.0)	0.2 (7.0)	0.183(4.0)	0.183(5.0)
breast-w	0.039 (4.5)	0.039(4.5)	0.036(3.0)	0.031(1.0)	0.043 (7.0)	0.033(2.0)	0.041(6.0)
sick	0.015 (1.0)	0.016(3.0)	0.018(4.0)	0.061(7.0)	0.016 (2.0)	0.018(5.0)	0.019(6.0)
tic-tac-toe	0.0090(1.0)	0.042(3.0)	0.081(5.0)	0.19 (7.0)	0.024 (2.0)	0.122(6.0)	0.053(4.0)
spambase	0.052 (4.0)	0.046(2.0)	0.046(1.0)	0.052(5.0)	0.059 (7.0)	0.049(3.0)	0.059(6.0)
cylinder-bands	0.219 (6.0)	0.187(3.0)	0.194(4.0)	0.154(1.0)	0.217 (5.0)	0.165(2.0)	0.381(7.0)
kr-vs-kp	0.009 (2.0)	0.009(3.0)	0.01 (4.0)	0.035(7.0)	0.006 (1.0)	0.031(6.0)	0.029(5.0)
avg. rank	3.38	2.98	2.68	3.5	5.3	4.45	5.73

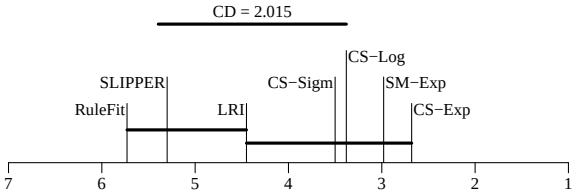

Fig. 3. Critical difference diagram

– ENDER: we chose the best four classifiers from the artificial data experiment, described in details in the previous section. For all classifiers, we set $M = 500$.

We used 20 binary classification problems, all taken from the UCI Repository [17]. Each test was performed using 10-fold cross validation (with exactly the same train/test splits for each classifier) and average 0-1 loss on testing folds was calculated. The results are shown in Table 1.

To compare multiple classifiers on multiple datasets, we follow [18], 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 35.636 which exceeds the critical value 12.592 (for confidence level 0.05), and 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 $CD = 2.015$ which means that algorithms with difference in average ranks greater than 2.015, are significantly different. In Figure 7.2 average ranks were marked on a line, and groups of the classifiers that are not significantly different were connected. This shows that all ENDER algorithms outperform SLIPPER and RuleFit, but are not significantly different from LRI. On the other hand, none of the three well-known rule ensemble algorithms (LRI, SLIPPER, RuleFit) is significantly better than any other.

The results confirm what we have already learned from the experiment on the artificial data. The choice of the loss function does not seem to be a critical issue in the learning process. The more important is the use of shrinkage, resampling and calculating the response of the rule on the whole dataset rather than on the subsample only. This can be noticed from the fact that SM-Exp is very similar to SLIPPER, but SM-Exp applies these techniques, what leads to much better results. We did not use in this experiment MLRules, because this algorithm is a variant of ENDER with logit loss.

8 Conclusions and Future Plans

We analyzed theoretically and empirically a general algorithm for learning an ensemble of decision rules. We found out that the loss function has a small impact on classifiers' performance. In turn, the minimization technique influences the form of the rules and their coverage. We also showed a simple technique consisting of shrinking base classifiers towards the priors and sampling from the training set when a single rule is built, but calculating a response of the rule on the whole training set. This technique offers an alternative to pruning and gives a significant improvement of accuracy. The introduced method outperformed three other well-known rule ensemble algorithms in our experiment.

An additional advantage of rules is their simplicity and interpretability. A question, however, arises whether an ensemble of 1000 rules is still interpretable. There exists an opinion that only a small set of rules can provide an insight into the analyzed phenomena. We believe that rule ensemble can still be used for interpretation purposes. One way is to follow the approach given in [10]. Another option is to sort the rules in the ensemble by some rule interestingness measure as it is done in the case of association rules. In this paper, we have shown that one can easily control the coverage of the rules. This feature can be utilized in searching for interesting rules. We postpone further analysis of this problem to our future work.

References

1. Michalski, R.S. In: A Theory and Methodology of Inductive Learning. Tioga Publishing, Palo Alto (1983) 83–129
2. Clark, P., Niblett, T.: The CN2 induction algorithm. Machine Learning **3** (1989) 261–283
3. Fürnkranz, J.: Separate-and-conquer rule learning. Artificial Intelligence Review **13**(1) (1996) 3–54
4. 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) (1997) 119–139
5. Hastie, T., Tibshirani, R., Friedman, J.H.: Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer (2003)
6. Mason, L., Baxter, J., Bartlett, P., Frean, M. In: Functional gradient techniques for combining hypotheses. MIT Press (1999) 33–58

7. Friedman, J.H.: Greedy function approximation: A gradient boosting machine. The Annals of Statistics **29**(5) (2001) 1189–1232
8. Cohen, W.W., Singer, Y.: A simple, fast, and effective rule learner. In: National Conference on Artificial Intelligence. (1999) 335–342
9. Weiss, S.M., Indurkhya, N.: Lightweight rule induction. In: International Conference on Machine Learning. (2000) 1135–1142
10. Friedman, J.H., Popescu, B.E.: Predictive learning via rule ensembles. Research report, Dept. of Statistics, Stanford University (2005)
11. Dembczyński, K., Kotłowski, W., Słowiński, R.: Maximum likelihood rules. In: International Conference on Machine Learning. (2008) 224–231
12. Friedman, J.H., Hastie, T., Tibshirani, R.: Additive logistic regression: a statistical view of boosting. Annals of Statistics (2000) 337–407
13. 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) (1998) 1651–1686
14. Friedman, J.H., Popescu, B.E.: Importance sampled learning ensembles. Research report, Dept. of Statistics, Stanford University (2003)
15. Schapire, R.E., Singer, Y.: Improved boosting algorithms using confidence-rated predictions. Machine Learning **37**(3) (1999) 297–336
16. Breiman, L.: Bagging predictors. Machine Learning **24**(2) (1996) 123–140
17. Asuncion, A., Newman, D.J.: UCI machine learning repository (2007)
18. Demšar, J.: Statistical comparisons of classifiers over multiple data sets. Journal of Machine Learning Research **7** (2006) 1–30