Maximum Likelihood Rule Ensembles

Krzysztof Dembczyński KDEMBCZYNSKI@CS.PUT.POZNAN.PL
Institute of Computing Science, Poznań University of Technology, Poznań, 60-965, Poland

Wojciech Kotłowski WKOTLOWSKI@CS.PUT.POZNAN.PL
Institute of Computing Science, Poznań University of Technology, Poznań, 60-965, Poland

Roman Słowiński RSLOWINSKI@CS.PUT.POZNAN.PL
Institute of Computing Science, Poznań University of Technology, Poznań, 60-965, Poland
Systems Research Institute, Polish Academy of Sciences, Warsaw, 01-447, Poland

Abstract

We propose a new rule induction algorithm for solving classification problems via probability estimation. The main advantage of decision rules is their simplicity and good interpretability. While the early approaches to rule induction were based on sequential covering, we follow an approach in which a single decision rule is treated as a base classifier in an ensemble. The ensemble is built by greedily minimizing the negative loglikelihood which results in estimating the class conditional probability distribution. The introduced approach is compared with other decision rule induction algorithms such as SLIPPER, LRI and RuleFit.

1. Introduction

Decision rule is a logical statement of the form: "if *condition* then *response*". It can be treated as a simple classifier that gives a constant response for the objects satisfying the condition part, and abstains from the response for all the other objects. Induction of decision rules has been widely considered in the early machine learning approaches (Michalski, 1983; Cohen, 1995; Fürnkranz, 1996), and rough set approaches to knowledge discovery (Stefanowski, 1998). The most popular algorithms were based on a sequential covering procedure (also known as separate-and-conquer approach). In this technique, a rule is learned which covers a part of the training examples, then examples are removed from the training set and the process is repeated until no examples remain.

Although it seems that decision (classification) trees are much more popular in data mining and machine learning applications, recently we are able to observe again a growing interest in decision rule models. As an example, let us mention such algorithms as RuleFit (Friedman & Popescu, 2005), SLIPPER (Cohen & Singer, 1999), Lightweight Rule Induction (LRI) (Weiss & Indurkhya, 2000). All these algorithms follow a specific iterative approach to decision rule generation by treating each decision rule as a subsidiary base classifier in the ensemble. This approach can be seen as a generalization of the 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 (RuleFit is an exception since it generates the trees first and then transforms them to rules). Each rule is fitted by concentrating on objects which were hardest to classify correctly by rules already present in the ensemble. All these algorithms can be explained within the framework of boosting (Freund & Schapire, 1997; Mason et al., 1999; Friedman et al., 2000) or forward stagewise additive modeling (FSAM) (Hastie et al., 2003), a greedy procedure for minimizing a loss function on the dataset.

The algorithm proposed in this paper, Maximum Likelihood Rule Ensembles (MLRules), benefits from the achievements in boosting machines (Freund & Schapire, 1997; Mason et al., 1999; Friedman et al., 2000; Friedman, 2001). Its main idea consists in rule induction by greedily minimizing the negative loglikelihood (also known as logit loss in binary classification case) to estimate the conditional class probability distribution. Minimization of such loss function with a

Appearing in *Proceedings of the 25th International Conference on Machine Learning*, Helsinki, Finland, 2008. Copyright 2008 by the author(s)/owner(s).

tree being a base classifier has already been used in LogitBoost (Friedman et al., 2000) and MART (Friedman, 2001), however, here we show a modified procedure, adapted to the case when the decision rule is a base classifier in the ensemble. In contrary to RuleFit (where trees are generated first), rules are generated directly; in contrary to SLIPPER and LRI, negative loglikelihood loss is used. Moreover, our approach is distinguished from other approaches to rule induction by the fact of estimating the class probability distribution instead of single classification and by using the same single measure (value of the negative loglikelihood) at all stages of the learning procedure: setting the best cuts (conditions), stopping the rule's growth and determining the response (weight) of the rule. We derive the algorithm for two optimization techniques, depending on whether we expand the loss function to the first order (fitting to the gradient) or to the second order (Newton steps). We report experiments showing the performance of MLRules and comparing them with the competitive rule ensemble methods.

The paper is organized as follows. In Section 2, the problems of classification is described. Section 3 presents a framework for learning rule ensembles. Section 4 is devoted to the problem of a single rule generation. In Section 5 we discuss the issue of convergence of the method, and we propose a modification to the main algorithm. Section 6 contains experimental results. The last section concludes the paper and outlines further research directions.

2. Problem Statement

In the classification problem, the aim is to predict the unknown class label $y \in \{1, \ldots, K\}$ of an object using known values of the attributes $\mathbf{x} = (x_1, x_2, \ldots, x_m)$. This is done by constructing a classification function $f(\mathbf{x})$ that predicts accurately the value of y. The accuracy of a single prediction is measured in terms of the loss function $L(y, f(\mathbf{x}))$, while the overall accuracy of the function $f(\mathbf{x})$ is measured by the expected loss (risk) over the data distribution $P(\mathbf{x}, y)$:

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

Since $P(\mathbf{x}, y)$ is unknown, the risk-minimizing function (Bayes classifier), $f^* = \arg\min_f \mathbb{E}[L(y, f(\mathbf{x}))]$, is also unknown. The learning procedure uses only a set of training examples $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$ to construct f to be a good approximation of f^*. Usually, it is performed by minimization of the empirical risk:

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

where function f is chosen from a restricted family of functions. The most commonly used loss function is 0-1 loss, $L_{0\text{-}1}(y, f(\mathbf{x})) = 1 - \delta_{y,f(\mathbf{x})}$, where $\delta_{ij} = 1$ if $i = j$, otherwise $\delta_{ij} = 0$. If the correct class is predicted, classification function is not penalized, otherwise the unit penalty is imposed. Bayes classifier has the following form:

$$f^*(\mathbf{x}) = \arg\min_{k \in \{1,\ldots,K\}} \Pr(y = k|\mathbf{x}). \tag{2}$$

The 0-1 loss has several drawbacks. Firstly, if we introduce unequal costs of misclassification, $f^*(\mathbf{x})$ does not longer have the form (2). Moreover, 0-1 loss is insensitive to the "confidence" of prediction: minimization of 0-1 loss results only in finding the most probable class, without estimating its probability. On the contrary, probability estimation provides us with the conditional class distribution $P(y|\mathbf{x})$, by which we can measure the prediction confidence. Moreover, all we need to obtain the Bayes classifier for *any* loss function is the conditional probability distribution. Here we consider the estimation of probabilities using the well-known maximum likelihood estimation (MLE) method. MLE can be stated as the empirical risk minimization by taking the negative logarithm of the conditional likelihood (negative log-likelihood) as the loss function:

$$\ell = \sum_{i=1}^{n} -\log P(y_i|\mathbf{x}_i). \tag{3}$$

We model probabilities $P(1|\mathbf{x}), \ldots, P(K|\mathbf{x})$ with a vector $\mathbf{f}(\mathbf{x}) = (f_1(\mathbf{x}), \ldots, f_K(\mathbf{x}))$ using the multinomial logistic transform:

$$P(y|\mathbf{x}) = \frac{e^{f_y(\mathbf{x})}}{\sum_{k=1}^{K} e^{f_k(\mathbf{x})}}. \tag{4}$$

Then (3) has the form:

$$\ell(\mathbf{f}) = \sum_{i=1}^{n} \log\left(\sum_{k=1}^{K} e^{f_k(\mathbf{x}_i)}\right) - f_{y_i}(\mathbf{x}_i). \tag{5}$$

This expression (with the exception that vector function $\mathbf{f}$ is used instead of scalar f) has the form of (1) if we identify $L(y_i, \mathbf{f}(\mathbf{x}_i)) = \log\left(\sum_{k=1}^{K} e^{f_k(\mathbf{x}_i)}\right) - f_{y_i}(\mathbf{x}_i)$. It is worth mentioning that the Bayes function $\mathbf{f}^*(\mathbf{x})$ is obtained by the inverse of (4).

3. Rules Ensembles

In this section, we describe the scheme of learning rule ensembles. Let X_j be the set of all possible values of attribute j. Condition part of the rule consists of a conjunction of elementary expressions of the 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, $j \in \{1, \dots, m\}$. We assume that in the case of ordered value sets, S_j has the form of the interval $[s_j, \infty)$ or $(-\infty, s_j]$ for some $s_j \in X_j$, so that the elementary expressions take the form $x_j \geq s_j$ or $x_j \leq s_j$. For nominal attributes, we consider elementary expressions of the form $x_j = s_j$ or $x_j \neq s_j$. Let Φ be the set of elementary expressions constituting the condition part of the rule and let $\Phi(\mathbf{x})$ be an indicator function equal to 1 if $\mathbf{x}$ satisfies the condition part of the rule (all elementary expressions in the condition part), otherwise $\Phi(\mathbf{x}) = 0$. We say that a rule *covers* an object $\mathbf{x}$, if $\Phi(\mathbf{x}) = 1$. The response of the rule is a vector $\boldsymbol{\alpha} \in \mathbb{R}^K$ assigned to the region defined by Φ. Therefore, we define a decision rule as:

$$\mathbf{r}(\mathbf{x}) = \boldsymbol{\alpha}\Phi(\mathbf{x}). \tag{6}$$

Notice that the decision rule takes only two values, $\mathbf{r}(\mathbf{x}) \in \{\boldsymbol{\alpha}, \mathbf{0}\}$, depending whether $\mathbf{x}$ satisfies the condition part or not. In this paper, we assume the classification function is a linear combination of M rules:

$$\mathbf{f}(\mathbf{x}) = \sum_{m=1}^{M} \mathbf{r}_m(\mathbf{x}). \tag{7}$$

Using (4), we can obtain conditional probabilities from (7). Moreover, from (4) it follows that $P(y|\mathbf{x})$ is a monotone function of $f_y(\mathbf{x})$. Therefore, from (2) we have that object $\mathbf{x}$ is classified to the class with the highest $f_k(\mathbf{x})$. Thus, combination (7) has very simple interpretation as a voting procedure: rules vote for each class k, and object $\mathbf{x}$ is classified to the class with the highest vote.

The construction of an optimal rules ensemble minimizing the negative loglikelihood (empirical risk) is a hard optimization problem. That is why we follow here a forward stagewise strategy (Hastie et al., 2003), i.e. the rules are added one by one, greedily minimizing the loss function:

$$\mathbf{r}_m = \arg\min_{\mathbf{r}} \ell(\mathbf{f}_{m-1} + \mathbf{r}) = \arg\min_{\Phi, \boldsymbol{\alpha}} \ell(\mathbf{f}_{m-1} + \Phi\boldsymbol{\alpha}), \tag{8}$$

where $\mathbf{r}_m$ is a rule obtained in the m-th iteration and $\mathbf{f}_{m-1}$ is the rule ensemble after $m-1$ iterations. It has been shown (Hastie et al., 2003) that "shrinking" the base classifier while adding it to the ensemble improves the prediction accuracy. That is why we set:

$$\mathbf{f}_m(\mathbf{x}) = \mathbf{f}_{m-1}(\mathbf{x}) + \nu \cdot \mathbf{r}_m(\mathbf{x}),$$

where $\nu \in (0, 1]$ is the shrinkage parameter, which constitutes a trade-off between accuracy and interpretability. Higher values ($\nu \sim 1$) produce smaller ensembles, while low values ($\nu \sim 0.1$) produce larger but more accurate ones.

4. Generation of a Single Rule

In this section, we describe how the algorithm generates single rules. In order to obtain a rule, one has to solve (8). The optimization procedure is still computationally hard. Therefore, we restrict analysis to the rules voting for only one class, so that the response of the rule has the form $\boldsymbol{\alpha} = \alpha\mathbf{v}$, where $\mathbf{v}$ is a vector with only one non-zero coordinate $v_k = 1$, for some $k = 1, \dots, K$, and α is a positive real value.

We propose two heuristic procedures for solving (8). The first, called gradient method (Mason et al., 1999), approximates $\ell(\mathbf{f} + \alpha\mathbf{v}\Phi)$ up to the first order with respect to α:

$$\ell(\mathbf{f} + \alpha\mathbf{v}\Phi) \simeq \ell(\mathbf{f}) + \alpha\ell'(\mathbf{f}, \mathbf{v}\Phi), \tag{9}$$

where

$$\ell'(\mathbf{f}, \mathbf{v}\Phi) = \left.\frac{\partial \ell(\mathbf{f} + \alpha\mathbf{v}\Phi)}{\partial \alpha}\right|_{\alpha=0}. \tag{10}$$

Since the first term in (9) is constant, minimization of the loss for *any* positive α is equivalent to minimization of the second term. Thus, if we define:

$$\mathcal{L}_m(\Phi) = \min_{\mathbf{v}} -\ell'(\mathbf{f}, \mathbf{v}\Phi) \tag{11}$$

(we remind, that there are only K possible vectors $\mathbf{v}$, so the arg min operation can be done by simply checking all K possibilities), then Φ_m can be obtained by minimizing $\mathcal{L}_m(\Phi)$.

The second heuristic, Newton method, approximates $\ell(\mathbf{f} + \alpha\mathbf{v}\Phi)$ up to the second order:

$$\ell(\mathbf{f} + \alpha\mathbf{v}\Phi) \simeq \ell(\mathbf{f}) + \alpha\ell'(\mathbf{f}, \mathbf{v}\Phi) + \frac{\alpha^2}{2}\ell''(\mathbf{f}, \mathbf{v}\Phi), \tag{12}$$

where $\ell'(\mathbf{f}, \mathbf{v}\Phi)$ is defined as before, and:

$$\ell''(\mathbf{f}, \mathbf{v}\Phi) = \left.\frac{\partial^2 \ell(\mathbf{f} + \alpha\mathbf{v}\Phi)}{\partial \alpha^2}\right|_{\alpha=0}. \tag{13}$$

Due to convexity of the loglikelihood, expression (12) is minimized by the Newton step:

$$\alpha = -\frac{\ell'(\mathbf{f}, \mathbf{v}\Phi)}{\ell''(\mathbf{f}, \mathbf{v}\Phi)}. \tag{14}$$

By substituting (14) into (12), and taking the square root, we get:

$$\mathcal{L}_m(\Phi) = \min_{\mathbf{v}} -\frac{\ell'(\mathbf{f}, \mathbf{v}\Phi)}{\sqrt{\ell''(\mathbf{f}, \mathbf{v}\Phi)}}, \tag{15}$$

and we can obtain Φ_m by minimizing $\mathcal{L}_m(\Phi)$.

Algorithm 1 MLRules

```
input: set of $n$ training examples $\{(y_i, \mathbf{x}_i)\}_1^n$,
       $M$ – number of decision rules.
output: rule ensemble $\{r_m(\mathbf{x})\}_1^M$.
$\mathbf{f}_0 := \boldsymbol{\alpha}_0$.
for $m = 1$ to $M$ do
    $\Phi_m(\mathbf{x}) = \arg\min_\Phi \mathcal{L}_m(\Phi)$
    $\boldsymbol{\alpha}_m = -\ell'(\mathbf{f}, \mathbf{v}\Phi)/\ell''(\mathbf{f}, \mathbf{v}\Phi)$
    $\mathbf{r}_m(\mathbf{x}) = \boldsymbol{\alpha}_m \Phi_m(\mathbf{x})$
    $\mathbf{f}_m(\mathbf{x}) = \mathbf{f}_{m-1}(\mathbf{x}) + \nu \mathbf{r}_m(\mathbf{x})$
end for
```

Expressions $\ell'(\mathbf{f}, \mathbf{v}\Phi)$ and $\ell''(\mathbf{f}, \mathbf{v})$ have a very simple form. Let $\mathbf{v}$ be such that $v_k = 1$. Then:

$$\ell'(\mathbf{f}, \mathbf{v}\Phi) = \sum_{\Phi(\mathbf{x}_i)=1} p_{ik} - \delta_{k,y_i}, \tag{16}$$

$$\ell''(\mathbf{f}, \mathbf{v}\Phi) = \sum_{\Phi(\mathbf{x}_i)=1} p_{ik}(1 - p_{ik}), \tag{17}$$

where $p_{ik} = P(k|\mathbf{x}_i)$ and $\delta_{i,j} = 1$ iff $i = j$. To calculate $\mathcal{L}_m(\Phi)$, these expressions must be obtained for each k.

What we still need for finding Φ_m using both gradient and Newton techniques, is a fast procedure for minimizing $\mathcal{L}_m(\Phi)$, regardless whether it is defined by (11) or (15). We propose the following simple iterative procedure: at the beginning, Φ_m is empty (no elementary expressions are specified) and we set $\mathcal{L}_m(\Phi) = 0$. In each step, an elementary expression $x_j \in S_j$ is added to Φ_m that minimizes $\mathcal{L}_m(\Phi)$ (if it exists). Such expression is searched by sequentially testing the elementary expressions, attribute by attribute. For ordered attributes, each expression of the form $x_j \geq s_j$ or $x_j \leq s_j$ is tested, for every $s_j \in X_j$; for nominal attributes, we test each expression of the form $x_j = s_j$ or $x_j \neq s_j$, for every $s_j \in X_j$. Adding new expressions is repeated until $\mathcal{L}_m(\Phi)$ cannot be decreased. We also simultaneously obtain $\mathbf{v}_m$, i.e. the value of $\mathbf{v}$ for which the minimum is reached in (11) or (15). Notice that since $\mathcal{L}_m(\Phi) = 0$ at the beginning, $\mathcal{L}_m(\Phi)$ must be strictly negative at the end, otherwise no rule will be generated. The procedure for finding optimal Φ is very fast and proved to be efficient in computational experiments. The ordered attributes can be sorted once before generating any rule. This procedure resembles the way the decision trees are generated. Here, we look, however, for only one path from the root to the leaf. Moreover, let us notice that a minimal value of $\mathcal{L}_m(\Phi)$ is a natural stop criterion in building a single rule and we do not use any other measures (e.g. impurity measures) for choosing the optimal cuts.

Having found Φ_m, we can obtain α_m by solving the following convex line-search problem:

$$\alpha_m = \arg\min_\alpha \ell(\mathbf{f} + \alpha \mathbf{v}_m \Phi_m). \tag{18}$$

To speed up the computations, we follow, however, simpler procedure and obtain α_m by the Newton step (14). The whole procedure for constructing the rule ensemble is presented as Algorithm 1. We call this procedure MLRules. Note that we start with $\mathbf{f}(\mathbf{x})$ equal to $\boldsymbol{\alpha_0}$, which is a "default rule" with fixed $\Phi(\mathbf{x}) \equiv 1$, while $\mathbf{v}_0$ and α_0 are obtained as usual. Since the response always indicates the majority class, such a rule serves as a default classification when no other rule covers a given object.

In our implementation of the algorithm, we employed the resampling technique (Friedman & Popescu, 2003), which is known to improve both accuracy and computational complexity. To obtain less correlated rules, we search for Φ_m, using (11) or (15), on a random subsample (drawn without replacement) of the training set of size $\eta < n$. Then, however, the response α_m is obtained using *all* of the training objects (including those objects, which have not been used to obtain Φ_m). This usually decreases $|\alpha_m|$, so it plays the role of a regularization method, and avoids overfitting the rule to the training set.

5. Extensions

In this section, we shortly discuss the problem of convergence and propose two simple extensions of the main algorithm.

5.1. Convergence

The procedure of obtaining α_m with the Newton step does not always decrease the empirical risk and does not guarantee the convergence of the algorithm. However, using a simple backtracking line-search is sufficient for convergence: we start with α_m obtained by the Newton step. If $\ell(\mathbf{f} + \alpha\mathbf{v}\Phi) < \ell(\mathbf{f})$, the procedure stops; otherwise repeat $\alpha_m := \alpha_m/2$ until the above condition is satisfied. This procedure ends after a finite number of steps, since from the definition of $\mathcal{L}_m$, either (11) or (15), it follows that $\mathcal{L}_m = 0$ if and only if $\ell'(\mathbf{f}, \mathbf{v}\Phi) = 0$, so if a rule is generated, $\mathcal{L}_m < 0$ and $\mathbf{v}\Phi$ is a descent direction. Therefore, ℓ is decreased in each iteration. Since ℓ is bounded from below, the procedure converges, i.e.: $\lim_{m\to\infty} \ell(\mathbf{f}_m) = \ell^\infty$. In the implementation of the algorithm we do not use such a procedure since the algorithm is stopped after M rounds anyway.

This raises the question, whether ℓ^∞ is the solution with the minimum achievable value of negative loglikelihood in the class of rule ensembles $\mathcal{F}$, i.e. if ℓ^∞ is equal to $\ell^* = \inf_{\mathbf{f}\in\mathcal{F}} \ell(\mathbf{f})$? The answer is negative because a greedy procedure is used to find the condition part of rule Φ. Then, even if a "descent direction" rule exists, the procedure may fail to find it (although the resampling strategy improves the procedure by randomly perturbing the training set in each iteration, which helps to avoid "local minima"). Nevertheless, this questions seems not to be of practical importance here, since we fix the maximal number of rules M. This is due to the empirical evidences showing that ensemble methods sometimes overfit on real-life data when the size of the ensemble is too large. In the next subsection, we describe another stopping condition, independent of the parameter M.

5.2. Avoiding overfitting

A decision rule has the form of m-dimensional rectangle. It can be shown, that the class of m-dimensional rectangles has Vapnik-Chervonenkis (VC) dimension equal to $2m$ and the VC dimension does not depend on the number of cuts. This is contrary to the tree classifier, for which the VC dimension grows to infinity with increasing number of cuts (nodes). Therefore, in case of tree ensembles, one usually specifies some constraints on tree complexity, e.g. maximal number of nodes, while in case of a rule ensemble no such constraints are necessary.

The theoretical results (Schapire et al., 1998) suggest that an ensemble with a simple base classifier (with low VC dimension) and high prediction confidence (margin) on the dataset generalizes well, regardless of the size of the ensemble. Nevertheless, we conducted the computational experiments which show that the performance of rule ensemble can deteriorate as the number of rules grows, especially for the problems with high noise level. Similar phenomenon has been observed for other boosting algorithms, in particular for AdaBoost (Mason et al., 1999; Friedman et al., 2000; Dietterich, 2000). Therefore, we propose a procedure for stopping the ensemble growth, based on the simple "holdout set" analysis.

Each rule is induced from the subsample of size $\eta < n$ without replacement. Thus, there are $n - \eta$ objects which do not take part in the induction procedure and can be used as a holdout set to estimate the quality of the induced rule. Since each rule votes for a single class, we calculate a simple 0-1 error (accuracy) of such a rule on the covered objects from the holdout set. A rule is *acceptable* if the holdout error is better (lower) than random guessing. Then, the stopping condition has the following form: in any p subsequent iterations at least q rules are not acceptable. Such "averaging" over the iterations removes variations and allows us to observe the longer-term behavior of rule acceptability. We set $p = 10$ and $q = 8$, and those values were obtained by noticing, that when the null hypothesis states that rules are not worse than random guessing, at least 8 unacceptable rules must be obtained in 10 trials to reject the null hypothesis in the binomial test with confidence level 0.05. Another possibility for stopping the ensemble growth is running the internal cross validation, but such procedure has not been used in the experiment due to computational complexity.

5.3. Ordinal classification

It is often the case that a meaningful order relation between class labels exists. For example, in recommender systems, users are often asked to evaluate items on five value ("stars") scale. Such problems are often referred to as ordinal classification problems. Here we show how the order relation can be taken into account in MLRules. Without loss of generality, we assume that the order between classes is concordant with the order between class labels coded as natural numbers $Y = \{1, \ldots, K\}$.

To capture the ordinal properties of Y, we only take into account rules voting for "at least" and "at most" class unions, where by "at least" class union we mean set $\{k, \ldots, K\}$ for some k, while by "at most" class union we mean $\{1, \ldots, k\}$. Such rules can be incorporated by considering the vectors $\mathbf{v}$ in the response of the rule to be of the form: $\mathbf{v} = \{-1, \ldots, -1, 1, \ldots, 1\}$ (vote for "at least" union) or $\mathbf{v} = \{1, \ldots, 1, -1, \ldots, -1\}$ (vote for "at most" union), so that the rule increases the probability of a class union, and not of a single class.

The whole algorithm remains the same, apart from the formulas (16) and (17), which now takes the form:

$$\ell'(\mathbf{f}, \mathbf{v}\Phi) = \sum_{\Phi(\mathbf{x}_i)=1} \sum_{k=1}^{K} v_k (p_{ik} - \delta_{k,y_i}), \qquad (19)$$

$$\ell''(\mathbf{f}, \mathbf{v}\Phi) = \sum_{\Phi(\mathbf{x}_i)=1} \sum_{1=t}^{K} v_k p_{ik} \left(1 - \sum_{k=1}^{K} v_k p_{ik}\right). \qquad (20)$$

The experimental verification of the usefulness of such rule representation is postponed for future research due to the lack of space.

6. Experimental Results

In this section, we show the results of the computational experiments on real datasets. First, we examine the behavior of the ensemble as the number of rules increases. Then, we compare our algorithm with existing approaches to rule induction.

6.1. Error curves

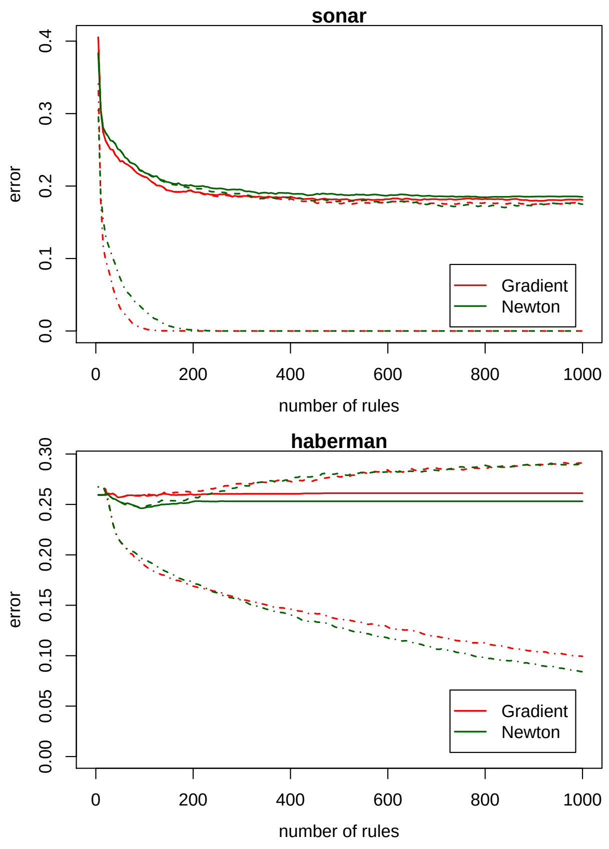

Figure 1. Train and test errors as functions of the ensemble size, obtained by splitting the data into train (66%) and test (33%) sets and averaging over 50 random splits. The lower lines (dashed-dotted) correspond to the train error, upper solid lines – to the test error with stopping condition described in Section 5.2, upper dashed line – test error when the stopping condition was not applied. Parameters of the MLRules are: $\nu = 0.1, \eta = 0.5n$

In Figure 6.1, we present the train and test error as a function of the ensemble size M for two real datasets, taken from the UCI Repository (Asuncion & Newman, 2007). On the *sonar* dataset, both ensembles (gradient and Newton) do not overfit and the test error decreases even if the number of rules reaches 1000; this is a rather typical situation. An atypical one can be found on the second dataset (*haberman*), where from some point, test error starts to increase. However, then a stopping condition described in Section 5.2 is satisfied which prevents rule ensemble from overfitting.

6.2. Comparison with other rule ensemble algorithms

To check the performance of MLRules on the real datasets, we compare them with three existing rule induction algorithms. SLIPPER (Cohen & Singer, 1999) was proposed within the AdaBoost reweighting scheme and uses an induction procedure which involves pruning. LRI (Weiss & Indurkhya, 2000) generates rules in the form of a DNF-formula and uses a specific reweighting scheme based on the cumulative errors. RuleFit (Friedman & Popescu, 2005) is based on FSAM framework (Hastie et al., 2003), but it uses the regression trees as base classifiers and then transforms them to rules. All three approaches are thus based on some boosting/reweighting strategy. According to our knowledge, RuleFit has not been compared with SLIPPER and LRI yet.

We used 35 files taken from UCI Repository (Asuncion & Newman, 2007), among which 20 files are binary classification tasks and 15 are multi-class tasks. We omit characteristics of the datasets due to lack of space. We tested four classifiers on each dataset (MLRules with gradient and Newton steps, LRI and SLIPPER) and RuleFit on binary datasets only (RuleFit does not handle multi-class case). We selected the following parameters for each method:

- SLIPPER: we set maximum number of iteration to 500, rest of parameters were set to default (we kept the internal cross validation, used to choose the optimal number of rules).
- LRI: According to (Weiss & Indurkhya, 2000), we set the rule length to 5 and froze feature after 50 rounds; we also 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 (Friedman & Popescu, 2005), we chose mixed rule-linear mode, set average tree size to 4, increased the number of trees to 500, and chose subsample size η as $\min\{0.5n, 100 + 6\sqrt{n}\}$.
- MLRules: We set $\eta = 0.5n, \nu = 0.1, M = 500$, but for the tree biggest datasets (*letter*, *optdigits*, *pendigits*) we increased M to 2000 (to compare, LRI had 26×200 rules for *letter*). Those parameters have not been optimized on the UCI data. We used artificial data to this end and due to the space limit we omit the characteristic of the data generating model.

Each test was performed using 10-fold cross validation

Table 1. Test errors and ranks (in parenthesis). MLRules.G and MLRules.N are MLRules with gradient and Newton method, respectively. Average ranks in the last row correspond to comparing LRI and MLRules on all 35 files.

Dataset	SLIPPER	LRI	RuleFit	MLRules.G	MLRules.N
		Binary-class Datasets			
haberman	0.268 (3.0)	0.275(5.0)	0.272(4.0)	0.262 (2.0)	0.249 (1.0)
breast-c	0.279 (3.0)	0.293(4.0)	0.297(5.0)	0.259 (1.0)	0.273 (2.0)
diabetes	0.254 (4.0)	0.254(3.0)	0.262(5.0)	0.247 (1.0)	0.253 (2.0)
credit-g	0.277 (5.0)	0.239(1.0)	0.259(3.0)	0.241 (2.0)	0.260 (4.0)
credit-a	0.170 (5.0)	0.122(1.0)	0.132(3.0)	0.133 (4.0)	0.130 (2.0)
ionosphere	0.065 (3.0)	0.068(4.0)	0.085(5.0)	0.060 (1.0)	0.063 (2.0)
colic	0.150 (4.0)	0.161(5.0)	0.147(3.0)	0.139 (2.0)	0.133 (1.0)
hepatitis	0.167 (2.0)	0.180(3.0)	0.194(4.0)	0.162 (1.0)	0.201 (5.0)
sonar	0.264 (5.0)	0.149(2.0)	0.197(4.0)	0.120 (1.0)	0.154 (3.0)
heart-statlog	0.233 (5.0)	0.196(4.0)	0.185(3.0)	0.167 (1.0)	0.174 (2.0)
liver-disorders	0.307 (5.0)	0.266(1.0)	0.307(4.0)	0.275 (2.0)	0.278 (3.0)
vote	0.050 (5.0)	0.039(3.0)	0.050(5.0)	0.034 (1.0)	0.037 (2.0)
heart-c	0.195 (5.0)	0.185(3.0)	0.189(4.0)	0.165 (2.0)	0.155 (1.0)
heart-h	0.200 (5.0)	0.183(3.0)	0.183(4.0)	0.180 (2.0)	0.170 (1.0)
breast-w	0.043 (5.0)	0.033(2.0)	0.041(4.0)	0.031 (1.0)	0.034 (3.0)
sick	0.016 (2.0)	0.018(4.0)	0.019(5.0)	0.016 (3.0)	0.012 (1.0)
tic-tac-toe	0.024 (2.0)	0.122(5.0)	0.053(3.0)	0.113 (4.0)	0.003 (1.0)
spambase	0.059 (5.0)	0.049(3.0)	0.059(4.0)	0.047 (2.0)	0.046 (1.0)
cylinder-bands	0.217 (4.0)	0.165(2.0)	0.381(5.0)	0.144 (1.0)	0.193 (3.0)
kr-vs-kp	0.006 (2.0)	0.031(5.0)	0.029(4.0)	0.010 (3.0)	0.005 (1.0)
avg. rank	3.9	3.15	4.05	1.85	2.05
		Multi-class Datasets			
anneal	0.018	0.007(3.0)	—	0.006 (1.5)	0.006 (1.5)
balance-scale	0.17	0.088(2.0)	—	0.078 (1.0)	0.091 (3.0)
ecoli	0.211	0.140(2.0)	—	0.149 (3.0)	0.140 (1.0)
glass	0.340	0.285(3.0)	—	0.244 (1.0)	0.248 (2.0)
iris	0.080	0.053(2.0)	—	0.053 (2.0)	0.053 (2.0)
letter	0.821	0.069(1.0)	—	0.137 (3.0)	0.088 (2.0)
segment	0.215	0.021(2.0)	—	0.029 (3.0)	0.020 (1.0)
soybean	0.505	0.413(3.0)	—	0.073 (2.0)	0.067 (1.0)
vehicle	0.301	0.210(1.0)	—	0.236 (3.0)	0.216 (2.0)
vowel	0.448	0.059(1.0)	—	0.148 (3.0)	0.104 (2.0)
car	0.045	0.054(2.0)	—	0.057 (3.0)	0.028 (1.0)
cmc	0.477	0.435(1.0)	—	0.437 (2.0)	0.439 (3.0)
dermatology	0.161	0.057(3.0)	—	0.019 (1.0)	0.024 (2.0)
optdigits	0.560	0.019(1.0)	—	0.026 (3.0)	0.021 (2.0)
pendigits	0.460	0.010(2.0)	—	0.014 (3.0)	0.010 (1.0)
avg. rank	—	2.26	—	1.9	1.84

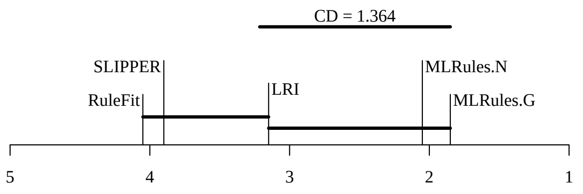

Figure 2. Critical difference diagram

(with exactly the same train/test splits for each classifier) and average 0-1 loss on the test set was calculated. The results are shown in Table 6.2.

We first restrict the analysis to binary-class problems only. To compare multiple classifiers on the multiple datasets, we follow Demšar (2006), and make the Friedman test, which uses ranks of each algorithm to check whether all the algorithms perform equally well (null hypothesis). Friedman statistics gives 33.28 which exceeds the critical value 9.488 (for confidence level 0.05), so we can reject the null hypothesis and state that classifiers are not equally good. Next, we proceed to a post-hoc analysis and calculate the *critical difference* (CD) according to the Nemeneyi statistics. We obtain $CD = 1.364$ which means that algorithms with difference in average ranks more than 1.364 are significantly different. In Figure 6.2 average ranks were marked on a line, and groups of classifiers that are not significantly different were connected. This shows that both MLRules algorithms are not significantly different to LRI, however they outperform both SLIPPER and RuleFit. On the other hand, none of three well-known rule ensemble algorithms (LRI, SLIPPER, RuleFit) is significantly better to any other.

The situation remains roughly the same if we compare the algorithms using all 35 datasets. We exclude RuleFit (it does not work with multi-class problems) and SLIPPER (its results are very poor, the worst almost every time[1]). Thus, we end up with 3 algorithms. Friedman statistics gives 3.53 which does not exceed the critical value 5.991, so that the null hypothesis cannot be rejected. Note that the difference in ranks decreased, mainly because LRI performs excellent on the largest datasets (letters and digits recognition).

It is interesting to check how much of the improvement in accuracy of MLRules comes from shrinkage, resampling and regularizing the rule response, because those techniques can also be simply incorporated to SLIPPER and LRI. We plan to investigate this issue in our future research.

7. Conclusions and Future Research

We proposed a new rule induction algorithm for solving classification problems, called MLRules, based on the maximum likelihood estimation method and using boosting strategy in rule induction. In contrary to previously considered algorithms, it estimates the conditional class probability distribution and therefore can work with any cost matrix for classification. We considered two optimization techniques, based on gradient and Newton steps, and introduced a stopping condition to avoid overfitting. The performance of MLRules was verified on a large collection of datasets, both binary- and multi-class. Our algorithm is competitive or outperforms the best existing approaches to rule induction.

We also suggested the way in which MLRules can capture the order between classes and therefore can solve the ordinal classification problems. We plan to verify this issue experimentally in the future.

[1]The SLIPPER software documentation says: "it is an *experiment* multi-class version" and "there is known bug which occasionally causes incorrect output", so we decided not to consider multi-class results.

References

Asuncion, A., & Newman, D. J. (2007). UCI machine learning repository.

Cohen, W. W. (1995). Fast effective rule induction. *ICML* (pp. 115–123).

Cohen, W. W., & Singer, Y. (1999). A simple, fast, and effective rule learner. *National Conference on Artificial Intelligence* (pp. 335–342).

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, *7*, 1–30.

Dietterich, T. G. (2000). An experimental comparison of three methods for constructing ensembles of decision trees: bagging, boosting, and randomization. *Machine Learning*, *40*, 139–158.

Freund, Y., & Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, *55*, 119–139.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, *29*, 1189–1232.

Friedman, J. H., Hastie, T., & Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting. *Annals of Statistics*, 337–407.

Friedman, J. H., & Popescu, B. E. (2003). *Importance sampled learning ensembles* (Technical Report). Dept. of Statistics, Stanford University.

Friedman, J. H., & Popescu, B. E. (2005). *Predictive learning via rule ensembles* (Technical Report). Dept. of Statistics, Stanford University.

Fürnkranz, J. (1996). Separate-and-conquer rule learning. *Artificial Intelligence Review*, *13*, 3–54.

Hastie, T., Tibshirani, R., & Friedman, J. H. (2003). *Elements of statistical learning: Data mining, inference, and prediction.* Springer.

Mason, L., Baxter, J., Bartlett, P., & Frean, M. (1999). *Functional gradient techniques for combining hypotheses*, 33–58. MIT Press.

Michalski, R. S. (1983). *A theory and methodology of inductive learning*, 83–129. Tioga Publishing.

Schapire, R. E., Freund, Y., Bartlett, P., & Lee, W. S. (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics*, *26*, 1651–1686.

Stefanowski, J. (1998). On rough set based approach to induction of decision rules. *Rough Set in Knowledge Discovering* (pp. 500–529). Physica Verlag.

Weiss, S. M., & Indurkhya, N. (2000). Lightweight rule induction. *ICML* (pp. 1135–1142).