An ℓ_1 Regularization Framework for Optimal Rule Combination

Yanjun Han and Jue Wang

Laboratory of Complex Systems and Intelligence Science,
Institute of Automation, Chinese Academy of Sciences
Beijing, 100190, P.R. China
{yanjun.han,jue.wang}@ia.ac.cn

Abstract. In this paper ℓ_1 regularization is introduced into relational learning to produce sparse rule combination. In other words, as few as possible rules are contained in the final rule set. Furthermore, we design a rule complexity penalty to encourage rules with fewer literals. The resulted optimization problem has to be formulated in an infinite dimensional space of horn clauses R_m associated with their corresponding complexity $\mathcal{C}_m$. It is proved that if a locally optimal rule is generated at each iteration, the final obtained rule set will be globally optimal. The proposed meta-algorithm is applicable to any single rule generator. We bring forward two algorithms, namely, ℓ_1FOIL and ℓ_1Progol. Empirical analysis is carried on ten real world tasks from bioinformatics and cheminformatics. The results demonstrate that our approach offers competitive prediction accuracy while the interpretability is straightforward.

Keywords: Rule Learning, ℓ_1 regularization, Path Following Algorithm.

1 Introduction

Relational Learning [3] deals with tasks in which observations are given in the form of fragments connected by relations. In general, these observations cannot be represented by vectors in the Euclidean Space $\mathbb{R}^n$. For instance, in chemistry, an observation (i.e., a molecule) is described by atoms connected by bonds. However, substructures extracted from atoms and bonds are responsible features for this domain rather than atoms and bonds themselves. Therefore, fragments are often integrated into rules (features) R_m on which the learner is constructed. The learner, which combines different rules together, is denoted as $y = f(R_1, R_2, ...R_M)$.

Due to its powerful expressive ability, Inductive Logic Programming (ILP) became the earliest approach for relational learning [15]. Most algorithms in ILP attempt to find a set of rules covering the given data set. These rules are iteratively generated and then combined by logical disjunction. Formally, $y = R_1 \vee R_2 \vee ... \vee R_M$.

However, there is no statistical guarantee for the generalization performance of the algorithms purely based on ILP. This motivated the research of integrating ILP with kernel methods. Usually these algorithms [7] consist of two steps: an appropriate kernel is designed with the assistance of ILP beforehand, and then a support vector machine (SVM) is trained based on the predefined kernel. In [11], another dynamic approach

W. Buntine et al. (Eds.): ECML PKDD 2009, Part I, LNAI 5781, pp. 501–516, 2009.

kFOIL was brought forward. In kFOIL, the rules are dynamically generated and combined in an SVM, i.e., $y = SVM(R_1, R_2, ...R_M)$. Its interpretability is embodied in the learned rule set. In addition, approaches have been proposed to embed ILP into probabilistic structures, such as the Bayesian network [12].

In essence, these approaches except kernel predefining possess the same mechanism: the rules are iteratively generated by ILP algorithms and then combined in a specific learner. The performance of an algorithm depends on whether it can choose the relevant rules. In current research, various theory revision techniques, heuristic tricks, and experts' experience [3] are utilized to seek the important rules. However, there is no theoretical and thus objective results to guarantee the global optimality of the obtained rule set. That's just the motivation of our work.

In this paper, we bring out an ℓ_1 regularization framework to produce sparse rule combination, which is applicable to any horn clauses generator. The clauses are dynamically generated in the training process, and we establish a one-to-one correspondence between clauses and features. Therefore, the observations are dynamically mapped into the feature space induced by horn clauses. Due to the ℓ_1 regularization term, the most relevant features can be automatically picked out [22]. We adopt the hinge loss as the loss function which has a sound statistical foundation [20]. In addition, for the hinge loss with ℓ_1 regularization, it is proved that there exists an efficient path following algorithm through which the entire solution path can be obtained [16]. Each section on the path corresponds to a rule set as an approximation to the true model. Then we can apply cross validation to choose the model with the highest accuracy. This greatly facilitates model selection in relational learning. To promote rules with lower complexity, we modify ℓ_1 penalty with the rule complexity weight. The resulted optimization problem is an ℓ_1 SVM in an infinite dimensional space, and we design a path following meta-algorithm to obtain its entire solution path. If at each iteration the generated clause is locally optimal, then the obtained rule set is proved to be globally optimal. Moreover, we propose a simple calibration procedure for the case that the generated clause is only suboptimal. There is an interesting interpretation from the viewpoint of kernel learning. The interpretability of our approach is satisfactory in that the obtained clauses are linearly combined, i.e., $y = f_1(R_1)+f_2(R_2)+...+f_M(R_M)$. We propose two algorithms, i.e. ℓ_1FOIL and ℓ_1Progol, based on the path following meta-algorithm. The results on real world tasks demonstrate that our approach has competitive prediction accuracy.

Throughout the paper we use the following notations: $\mathcal{M}$ is the index set of the dimensions in the feature space. $\phi(x) \in \mathbb{R}^{\mathcal{M}}$ is the mapping of x into the feature space ($\phi_m(x) \in \mathbb{R}$ is the "*m-th*"coordinate of $\phi(x)$ for $m \in \mathcal{M}$). For a certain relational learning problem, $\mathcal{H}$ is the corresponding space of horn clauses. It is not hard to show that $|\mathcal{H}| \leq \aleph_0$ (finite or countably infinite). In our setting, there is a one-to-one correspondence between $\mathcal{H}$ and $\mathcal{M}$. Therefore, the *j-th* clause $h_j \in \mathcal{H}$ can be denoted by a $|\mathcal{M}|-$ length vector $\boldsymbol{e}_j$ whose *j-th* element is 1 and all other elements are 0. However, $\mathcal{H}$ is indexed dynamically in the rule constructing process rather than indexed beforehand.

The remainder of the paper is organized as follows: ℓ_1 regularization is introduced into relational learning in Section 2. In Section 3 a path following algorithm which generates the entire solution path is presented and the resulted rule set is proved to be optimal in the infinite space of horn clauses under certain condition. We interpret our

approach from kernel learning viewpoint in section 4. In Section 5, the algorithm is evaluated on ten real world data sets. Finally, conclusions are drawn in Section 6.

2 Rule Complexity Penalty and Infinite Dimensional Rule Space

An ideal relational learning algorithm should possess the following two features: first, the relevant rules can be chosen from the vast ocean of candidates; second, the obtained rules are combined in an appropriate manner. The space of horn clauses for a certain relational learning task with n observations can be divided into 2^n equivalence classes, each of which corresponds to a set of rules whose output on the given observations are exactly the same. It seems that each equivalence class could be represented by any individual rule in it. However, this is far from true in that two rules with the same output on the training set may differ greatly on generalization performance. As will be analyzed below, it is more appropriate to differentiate the rules from the same equivalence class and formulate the optimal rule combining problem in infinite dimensional spaces. First of all, the relationship between features and rules has to be established.

2.1 The Correspondence between Features and Rules

Given: the background theory B, in the form of a set of horn clauses, i.e., clauses of the form $h \leftarrow b_1, ..., b_k$ where h and b_j are logical atoms. Background Knowledge depicts the basic facts and constraints in the given domain; a set of observations $D = \{x_i\}_{i=1}^n$, each of which is in the form of the ground facts and the corresponding label set is $\{y_i\}_{i=1}^n$; a set of rules $\{R_m\}_{m=1}^M \subset \mathcal{H}$, R_m are horn clauses.

The feature space is constructed as follows: If an observation x_i satisfies the rule R_m, its corresponding feature value $\phi_m(x_i)$ is set to 1, else to 0. Formally:

$$\phi_m(x_i) = \begin{cases} 1 & B \cup \{R_m\} \models x_i \\ 0 & B \cup \{R_m\} \not\models x_i \end{cases} \tag{1}$$

Therefore, there is a one-to-one correspondence between features and rules. Therefore, we will refer to them without differentiation. Nevertheless, the feature space is constructed dynamically in the learning process rather than determined beforehand.

2.2 ℓ_1 Penalty and Rule Complexity Penalty

For interpretability, the features are often linearly combined

$$f = \sum_{m \in \mathcal{M}} \beta_m \phi_m(x) + b, \quad |\mathcal{M}| \leq \aleph_0. \tag{2}$$

On the other hand, the ℓ_1 regularization is very popular in statistics and machine learning [17,22] because it encourages sparsity. This statistical terminology means that in the obtained linear model only the coefficients of a small number of features will be nonzero while the coefficients of other features will be exactly zero. Moreover, some theoretical research [9,22] indicate that under certain conditions the true model can be

obtained by algorithms with ℓ_1 regularization. For linear rule combination, this implies that it is possible to pick out the rules involved in the underlying model through ℓ_1 regularization. Consequently, the problem is formulated as follows:

$$\min_{\boldsymbol{\beta},b} \sum_{i=1}^{n} L(y_i, \boldsymbol{\beta}^T \boldsymbol{\phi}(x_i) + b) \quad s.t.\ \|\boldsymbol{\beta}\|_1 \leq t \tag{3}$$

where the loss function L is convex, $\|\boldsymbol{\beta}\|_1 = \sum_{m \in \mathcal{M}} |\beta_m|$ and $|\mathcal{M}| \leq \aleph_0$. The above constraint form is equivalent to the following regularization form:

$$\min_{\boldsymbol{\beta},b} \sum_{i=1}^{n} L(y_i, \boldsymbol{\beta}^T \boldsymbol{\phi}(x_i) + b) + \lambda \|\boldsymbol{\beta}\|_1 \tag{4}$$

There is a one-to-one correspondence between λ and t [16]. In this paper, we will use the constraint form (3) which is more convenient for algorithm designing.

Features ϕ_m are generated dynamically in the training process. There are 2^n choices for ϕ_m, and thus it seems that finite space is enough to analyze the optimal rule combining problem (3). However, due to the characteristic of relational learning, an infinite dimensional space is more appropriate for this problem. There are three reasons: firstly, the rules in the same equivalence class may have different complexity (e.g,: an equivalence class may contains the rules of the form $\{w_1x_1 + w_2x_2 > 0\}, R_1 \rightarrow pos$, and an appropriate complexity measure for this kind of rules is $\|w\|_1 + 2$ in which 2 stands for the number of literals. Therefore, the complexity for these rules are real numbers and thus have potentially infinite values and rules with different complexity should be distinguished from each other; secondly, the rules in the same equivalence class may have different numbers of literals and shorter rules with less redundancy are preferred (e.g.: rule generators may yield rules of infinite length due to their rigidity, such as $x > 0, x > -1, x > -2, \cdots \rightarrow Positive(x)$); finally, the rules are dynamically generated and thus it is more convenient to assume they are from an infinite repository.

To encourage simpler rules, we substitute the ℓ_1 penalty $\|\boldsymbol{\beta}\|_1$ in (3) with rule complexity penalty $\|\boldsymbol{\mathcal{C}} \odot \boldsymbol{\beta}\|_1$, where $\odot$ is the element-wise product of two vectors and $\mathcal{C}_m$ is the complexity of rule R_m. The resulted optimization problem is as follow:

$$\min_{\boldsymbol{\beta},b} \sum_{i=1}^{n} L(y_i, \boldsymbol{\beta}^T \boldsymbol{\phi}(x_i) + b) \quad s.t.\ \|\boldsymbol{\mathcal{C}} \odot \boldsymbol{\beta}\|_1 = \sum_{m \in \mathcal{M}} |\mathcal{C}_m \beta_m| \leq t \tag{5}$$

Furthmore, (5) can be rewritten as follow:

$$\min_{\boldsymbol{\beta},b} \sum_{i=1}^{n} L(y_i, \boldsymbol{\beta}^T \tilde{\boldsymbol{\phi}}(x_i) + b) \quad s.t.\ \|\boldsymbol{\beta}\|_1 \leq t \tag{6}$$

where $\tilde{\phi}_m(x_i) = \phi_m(x_i)/\mathcal{C}_m$.

Therefore, optimization problems (3) and (5) are of the identical type. In this work, we measure the complexity of a clause by the number of literals, i.e., $\mathcal{C}_m = \mathcal{N}_m$.

The effect of the rule complexity penalty consists of two parts: the output of rules are weighted according to its complexity and ℓ_1 regularization is performed on the weighted feature space. Moreover, for rules in each equivalence class their complexity may have infinite different values, and thus (6) is an infinite dimensional optimization problem.

2.3 The Optimality in Infinite Dimensional Rule Space and ℓ_1SVM

In [17] it is proved that the infinite dimensional problem (6) has a finite dimensional optimal solution. We restate and explain relevant theorems in [17] here.

Theorem 1. (Caratheodory's Convex Hull Theorem) *Let A be a finite set of points in $\mathbb{R}^n$. Then every $x \in co(A)$ can be expressed as a convex combination of at most $n+1$ points of A, where $co(A)$ is the set of all convex combinations of points in A.*

Theorem 1 can be readily extended to the case in which A is infinite. This theorem implies that for a given relational learning problem with n observations, it is possible to represent these observations with at most $n+1$ rules, as follows:

Theorem 2. (A sufficient condition for the existence of a sparse solution) *If the set $D = \{\phi_m(x) : m \in \mathcal{M}\} \subset \mathbb{R}^n$ is compact, then problem (6) has an optimal solution. Moreover, there exists an optimal solution supported on at most $n+1$ features in $\mathcal{M}$.*

Furthermore, a criterion is given to judge whether an $n+1$ dimensional solution is an optimal solution to (6).

Theorem 3. (The criterion for judging the optimality of a finite-supported solution) *If an optimal solution to (6) exists, and $\hat{\boldsymbol{\beta}}$ is a finite-supported candidate solution such that $\exists \mathcal{A} \subset \mathcal{M}, |\mathcal{A}| < \infty$ and $supp(\hat{\boldsymbol{\beta}}) = \mathcal{A}$. We can test its optimality using the following criterion:*
$\hat{\boldsymbol{\beta}}$ is optimal solution to (6) $\Leftrightarrow \forall \mathcal{B}$ s.t. $\mathcal{A} \subseteq \mathcal{B}, |\mathcal{B}| < \infty$, $\hat{\boldsymbol{\beta}}$ is optimal solution to:

$$\min_{\boldsymbol{\beta}} \sum_{i=1}^{n} L(y_i, \sum_{m \in \mathcal{B}} \beta_m \phi_m(x_i)) \quad s.t. \|\boldsymbol{\beta}\|_1 \leq \sum_{m \in \mathcal{A}} \hat{\beta}_m$$

This theorem implies that it suffices to show that a finite solution is optimal for any finite sub-problems containing it in order to prove its optimality for the original infinite problem. In the next section, we will utilize these theorems to prove that our algorithm indeed generate the optimal rule set.

In our work, the hinge loss is employed as the loss function L in (6), because there is sound theoretical foundation for SVM to guarantee its generalization performance [20]. Therefore, the obtained rules are linearly combined in the following ℓ_1SVM [24]:

$$\min_{\boldsymbol{\beta}, b} \sum_{i=1}^{n} \max(0, 1 - y_i(\boldsymbol{\beta}^T \tilde{\boldsymbol{\phi}}(x_i) + b)) \quad s.t. \ \|\boldsymbol{\beta}\|_1 \leq t. \tag{7}$$

where $\tilde{\boldsymbol{\phi}}(x_i) = \frac{1}{\boldsymbol{\mathcal{N}}} \odot \boldsymbol{\phi}(\boldsymbol{x_i})$, $\mathcal{N}_m$ is the number of literals in R_m.

It is proved in [23] that in finite case the solution to (7) has an amazing piecewise linearity. Namely, $\boldsymbol{\beta}$ is a piecewise linear function of t. When t varies from zero to

infinity, the entire path of β is generated. Each section of the path corresponds to a certain model $\hat{\beta}^*$, and this property greatly facilitates model selection. In this paper, we generalize ℓ_1SVM from finite feature spaces to the infinite space of horn clauses and design a similar path following algorithm to obtain the approximately optimal rule set.

3 The Algorithm

First of all, a nested algorithm integrating any rule generator and ℓ_1SVM is presented, and then we prove that an optimal rule set in the space of horn clauses can be generated by our algorithm if the optimal rule for each step is obtained.

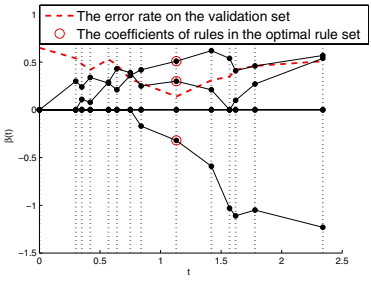


Fig. 1. The solution $\hat{\beta}(t)$ as a function of t

Table 1. Details for optimization in step 2.5 and 2.6

2.5 $P(R): \min_g -\sum_{i\in\mathcal{L}} y_i(\sum_{j\in\mathcal{A}} g_j \frac{\phi_j(x_i)}{\mathcal{N}_j} + g_R\phi_R(x_i)/\mathcal{N}_R)$

$$\text{s.t.} \begin{cases} \sum_{j\in\mathcal{A}} g_j \frac{\phi_j(x_i)}{\mathcal{N}_j} + g_R \frac{\phi_R(x_i)}{\mathcal{N}_R} = 0, \\ \quad \text{for } i \in \mathcal{E} \\ \sum_{j\in\mathcal{A}} sign(\beta_j) g_j + |g_R| = 1 \\ g_j = 0, for j \notin \mathcal{A} \cup \{\mathcal{I}_R\} \end{cases}$$

2.6 $Q(e): \min_g -\sum_{\mathcal{L}} y_i g^T \phi(x_i)/\mathcal{N}$

$$\text{s.t.} \begin{cases} \sum_{j\in\mathcal{A}} g_j \frac{\phi_j(x_i)}{\mathcal{N}_j} = 0 \text{ ,for } i \in \mathcal{E} \setminus \{e\} \\ \sum_{j\in\mathcal{A}} sign(\beta_j) g_j = 1 \\ g_j = 0, \text{for } j \notin \mathcal{A} \end{cases}$$

3.1 The Path Following Algorithm

There are two essential differences between the scenarios confronted in the seminal paper of ℓ_1SVM and here: first, the feature space induced by the horn clauses is potentially infinite rather than finite; second, the feature space here have to be dynamically constructed during the learning process rather than fixed beforehand. Therefore, the path following algorithm presented in [24] should be modified accordingly.

To describe the algorithm for finding the solution path, some concepts and definitions should be given first of all. A feature ϕ_m is called "active" when its corresponding coefficient β_m is nonzero. Features enter the active feature set $\mathcal{A} = \{j : \beta_j \neq 0\}$ successively when the value of t increases. When t is small, only the most relevant features can enter the active set $\mathcal{A}$; when t is large, $\mathcal{A}$ contains most of the features. The solution has a favorable piecewise linear property (Figure 1), i.e., $\forall m \in \mathcal{A} = \{j \mid \hat{\beta}_j \neq 0\}$,

Table 2. The Main Algorithm

1. **Initialize:**
   $\mathcal{HG}$: a Horn clause Generator. $\mathcal{N}_R$: the number of literals in R.
   The rule set $\mathcal{R} = \emptyset$,
   $R^* = \text{argmin}_R \Delta L(\phi_R) = \text{argmin}_{\mathcal{HG} \to R} - \sum_{\mathcal{L}} y_i \phi_R(x_i)/\mathcal{N}_R$, $\mathcal{R} = \{R^*\}$
   $\mathcal{A} = \{\mathcal{I}_{R^*}\}$, the index of R^* in the whole set of horn clauses.
   $g_j = 1, j \in \mathcal{A}$; $g_j = 0, j \notin \mathcal{A}$
   $\Delta L^* = \Delta L(R^*)$. $m = 0$, $\mathcal{A}^m = \mathcal{A}$, $P^m(R) = -\sum_{\mathcal{L}} y_i \phi_R(x_i)/\mathcal{N}_R$, $\beta^m = \mathbf{0}$, $m = m + 1$

2. **While** $\Delta L^* \neq 0$
   2.1 $d_1 = \min\{d > 0 : (\beta + dg)_j = 0, j \in \mathcal{A}\}$
   2.2 $d_2 = \min\{d > 0 : 1 - y_i(\beta + dg)^T \phi(x_i)/\mathcal{N} = 0, i = 1, ..., n\}$
   2.3 set $d = min(d_1, d_2)$. $\beta = \beta + dg$.
   2.4 **If** $d = d_1$, **then** remove the variable attaining 0 from $\mathcal{A}$,
   If $d = d_2$, **then** add the observation entering the elbow to $\mathcal{E}$.
   2.5 $\mathcal{HG}$ is guided by : $\min_{\mathcal{HG} \to R} P(R)$, $P(R)$ is detailed in Table 1.
   The solution is (R^*, g^*), set $\phi(\cdot) = \{\phi_j(\cdot)\}_{j \in \mathcal{A}} \cup \phi_{R^*}(\cdot)$,
   $\Delta L_1 = -\sum_{\mathcal{L}} y_i g^{*T} \phi(x_i)/\mathcal{N}$. ($\mathcal{N}_m$: the number of literals in R_m).
   2.6 $\min_{e \in \mathcal{E}} Q(e)$, $Q(e)$ is detailed in Table 1.
   The solution is (e^*, g^*), $\Delta L_2 = -\sum_{\mathcal{L}} y_i g^{*T} \phi(x_i)/\mathcal{N}$.
   2.7 Let $\Delta L^* = min\{\Delta L_1, \Delta L_2\}$,
   If $\Delta L^* = \Delta L_1$, **then**
   invoke the **Calibration Procedure** (Section 3.3):
   $\mathcal{A}^m = \mathcal{A}$, $P^m(\cdot) = P(\cdot)$ (step 2.5), $\beta^m = \beta$, $R^m = R^*$, $m = m + 1$.
   Update g and $\mathcal{A} \leftarrow \mathcal{A} \cup \{\mathcal{I}_{R^*}\}$. $\mathcal{R}_{\mathcal{A}} = \mathcal{R}_{\mathcal{A}} \cup \{R^*\}$
   If $\Delta L^* = \Delta L_2$, **then** update g and $\mathcal{E} \leftarrow \mathcal{E} \setminus \{e^*\}$, $\mathcal{L} \leftarrow \mathcal{L} \cup \{e^*\}$.
   If $\Delta L^* \leq 0$, **then** set $\Delta L^* = 0$.
   2.8 **Return** to 2.

$\hat{\beta}_m(t)$ is a piecewise linear function of t. Piecewise linear property only depends on the functional form of the loss function and the regularization term [16], and thus it still holds in the infinite dimensional space. Since the hinge loss $L(v) = max(0, 1 - v)$ is non-smooth, the dataset has to be divided into three non-intersecting sets in order to carry out analysis, as follows:

$\mathcal{E} = \{i : 1 - y_i \beta^T \phi(x_i) = 0\}$, elbow set of observations;
$\mathcal{L} = \{i : 1 - y_i \beta^T \phi(x_i) > 0\}$, left of the elbow set;
$\mathcal{R} = \{i : 1 - y_i \beta^T \phi(x_i) < 0\}$, right of the elbow set (has no use in analysis);

Due to the nonsmoothness of the loss function and the ℓ_1 constraint, it is impossible to guide the optimization problem directly via gradient information. Intead, we have to solve the following optimization to get the steepest descent direction for L:

$$\begin{aligned} \boldsymbol{g}_{\boldsymbol{\beta}} = \underset{\boldsymbol{d}_{\boldsymbol{\beta}}}{\text{argmin}} \quad & \Delta L(\epsilon \boldsymbol{d}_{\boldsymbol{\beta}}) = L(\boldsymbol{\beta} + \epsilon \boldsymbol{d}_{\boldsymbol{\beta}}) - L(\boldsymbol{\beta}) = -\sum_{\mathcal{L}} y_i \epsilon \boldsymbol{d}_{\boldsymbol{\beta}}^T \boldsymbol{\phi}(x_i) \\ s.t. \quad & \epsilon \boldsymbol{d}_{\boldsymbol{\beta}}^T \boldsymbol{\phi}(x_i) = 0 \text{ for } i \in \mathcal{E}, \|\boldsymbol{d}_{\boldsymbol{\beta}}\| = 1 \end{aligned} \tag{8}$$

"$\boldsymbol{g}$" and "$\boldsymbol{d}$" stand for gradient and descent direction, respectively. $\boldsymbol{g}_{\boldsymbol{\beta}}$ is the direction in which the loss function decreases fastest per unit increase of the ℓ_1 norm of the

coefficient vector. If ϵ is a sufficiently small positive number, the observations in $\mathcal{R}$ and $\mathcal{L}$ still remains in their respective set when the coefficient vector grows from $\boldsymbol{\beta}$ to $\boldsymbol{\beta} + \epsilon \boldsymbol{d}_{\boldsymbol{\beta}}$. However, an observation may leave $\mathcal{E}$ and enter $\mathcal{L}$, and thus it is impossible to determine the incremental of the loss function correpsonding to the incremental of the coefficient vector $\epsilon \boldsymbol{d}_{\boldsymbol{\beta}}$. Therefore, we add a contraint (the first constraint in (8)) to keep $\mathcal{E}$ unchanged. Furthermore, (8) can be simplified to the following form:

$$g_{\boldsymbol{\beta}} = \underset{\boldsymbol{d}_{\boldsymbol{\beta}}}{\operatorname{argmin}} - \sum_{\mathcal{L}} y_i \boldsymbol{d}_{\boldsymbol{\beta}}^T \boldsymbol{\phi}(x_i) \quad s.t. \quad \boldsymbol{d}_{\boldsymbol{\beta}}^T \boldsymbol{\phi}(x_i) = 0 \text{ for } i \in \mathcal{E}, \|\boldsymbol{d}_{\boldsymbol{\beta}}\| = 1 \tag{9}$$

(9) is the basis for the optimization problem of step 2.5 and 2.6 (Table 2).

First of all, the rule generator algorithm is invoked to generate a feature (rule) that can decrease the loss the fastest per unit increase of ℓ_1 norm of the coefficient vector. The solution moves along that direction until one of the following two events happens: 1. A coefficient hits the non-differentiable points of the penalty, i.e., β_j becomes zero from nonzero for some j. Hence the corresponding rule is removed from $\mathcal{A}$. (step 2.1) 2. An observation hits the non-differentiable point of the loss, i.e., $1 - y_i \boldsymbol{\beta}^T \boldsymbol{\phi}(x_i)$ becomes zero from nonzero for some i. Hence this observation is added to $\mathcal{E}$ (step 2.2).

Then there are three types of actions one can take: 1. invoke the rule generator to yield a new rule and add it to $\mathcal{A}$. (step 2.5) 2. remove an observation from $\mathcal{E}$. (step 2.6) 3. do nothing (can be merged with the first action). The action that can cause the fastest decrease in the loss per unit increase of the ℓ_1 norm of the coefficient will be chosen.

3.2 The Optimality of the Algorithm

Now we will prove that if at each iteration an optimal rule for current step is generated, then the obtained rule set is an optimal solution to infinite dimensional optimization problem (6). This is an amazing property in that generally the global optimality of the rule set cannot be guaranteed just by the local optimality of single rules due to the greedy nature of rule generating process. This characteristic distinguishes our approach from existing rule combining methods.

Theorem 4. *Assume at any iteration of the algorithm, an optimal rule for step 1 and step 2.5 (**Table 2**) is obtained. For a certain iteration, assume we are after step 2.2 and the current rule set is $\mathcal{R}_{\mathcal{A}}$. Then the corresponding finitely-supported solution $\boldsymbol{\beta}_{\mathcal{A}}$ is an optimal solution to (7) in infinite dimensional rule space with $t = \|\boldsymbol{\beta}_{\mathcal{A}}\|_1$. Namely, $\mathcal{R}_{\mathcal{A}}$ is an optimal rule set with the constraint $t = \|\boldsymbol{\beta}_{\mathcal{A}}\|_1$.*

Proof: Given a relational learning task and n observations $X = \{x_i\}_{i=1}^n$, the set of horn clauses can be divided into 2^n equivalence classes $H_p, p = 1, \cdots, 2^n$ according to the output on X. The corresponding feature value vector $\phi_p(X)$ is constructed as in (1) and the index set is $\mathcal{I}_p$. In this work, we measure the complexity of a rule R by its number of literals, i.e., $\mathcal{C} = \mathcal{N}_R$. $\forall h \in H_p$, if $\mathcal{N}_h$ is finite, then the feature vector set $D_p = \{\tilde{\phi}_i = \phi_p(X)/\mathcal{N}_{h_i}, i \in \mathcal{I}_p\}$ is finite. If $\exists h, \mathcal{N}_h$ is infinite, then D_p is a countable infinite set with a unique limit point $\mathbf{0}$ (a $n \times 1$ vector). Therefore, the overall feature vector set $D = \bigcup_{p=1}^{2^n} D_p$ is a bounded set with unique limit point $\mathbf{0}$. We construct a new feature vector set $D' = D \cup \{\mathbf{0}\}$. It is easy to show that the solution set for (7) with D'

equals that with D. Moreover, $D^{'}$ is a bounded closed set, and thus compact. According to Theorem 2, there is an optimal solution supported on at most $n+1$ features in $D^{'}$. Due to the equivalence between optimizing with D and $D^{'}$, this also holds for D. Next we will prove the optimality of the finitely-supported solution at any iteration. Assume at any iteration , an optimal rule for step 1 and step 2.5 (**Table 2**) is obtained. Therefore, there is no need to perform calibration at step 2.7. For finite D, the optimality of the algorithm has been proved in [23]. For infinite D, since the finite feature set result implies that for any finite $\mathcal{B}$ such that $\mathcal{A} \subseteq \mathcal{B} \subseteq \mathcal{M}$, the finitely-supported solution $\hat{\boldsymbol{\beta}}_{\mathcal{A}}$ is also an optimal solution in feature set $\mathcal{B}$ given the restriction $\|\boldsymbol{\beta}_{\mathcal{B}}\|_1 \leq \|\hat{\boldsymbol{\beta}}_{\mathcal{A}}\|_1$. Theorem 3 and finite feature set result combined complete the proof. □

3.3 Suboptimality and Calibration Procedure

In practice the exact optimization in step 2.5 is prohibitive because the space of horn clauses is exponential to the number of observations or infinite. Therefore, at each step 2.5 a suboptimal rule is chosen instead of the theoretically optimal one. It is still not clear that to which extent the performance of the algorithm is influenced by this gap between theory and reality. We conjecture that this gap does not significantly impair the performance of our algorithm, since the difference between the optimal rule and the suboptimal rule lies in that the former corresponds to the steepest descent direction while the latter only corresponds to ordinary descent direction, and for convex objective function, an algorithm proceeds in an ordinary descent direction at each iteration can still converge at a slower rate. In experimental evaluation, we explored the impact of this gap on our algorithms' performance, and the results support our conjecture. We will carry out detailed theoretical analysis in the future work.

Nevertheless, there is a situation which has to be avoided: the "contradiction" on the solution path. Formally, for two iteration steps $m_2 > m_1$, rules selected at step 2.5 are R_{m_2} and R_{m_1} respectively, if R_{m_2} is better for the optimization problem P^{m_1} than R^{m_1}, then we say that an contradiction happens. A simple remedy is that once a new rule is generated, it is substituted into previous P_m to examine whether an contradiction exists, and algorithm is restarted at the earliest step when the contradiction happens.

3.4 Analysis of Computational Complexity

The computational cost of the algorithm consists of three parts: the first part is brought by the rule generator when searching for the optimal rule at step 2.5, the second part

Table 3. The Calibration Procedure

```
for n = 0 : (m − 1)
  if P^n(R*) < P^n(R^n)
    if n = 0, goto step 1
    else set m = n − 1, A = A^{n−1}, β = β^{n−1}, goto step 2.1
    end if
  end if
end for
```

is resulted in by the linear programming at step 2.5 and 2.6, and the third part is due to the computation for the entire solution path. The following notations are used in analysis: for *m-th* iteration, $|\mathcal{E}| = q$ and the size of current $\mathcal{A}$ is s_a. The number of the rules that have ever entered the active rule set $\mathcal{A}$ during the overall iterations is p, the complexity of the rule generator is $\mathcal{O}(C_f)$, and the number of invoking the rule generator to generate an approximate rule at step 2.5 is C_s.

The complexity for solving the inner linear programming at step 2.5 and 2.6 is $\mathcal{O}(s_a^3)$ [13]. If the training data is separable, then the number of joints on the path can be assumed to be $\mathcal{O}(n)$. Therefore, the overall complexity is $\mathcal{O}(((C_f + s_a^3)C_s + q \cdot s_a^3) \cdot n)$. In experiment, we find $C_f \propto n^2$, $C_s < 40$ and $p < n$. Since $q \leq min(n, p)$ and $s_a \leq p$, then the worst case complexity is $\mathcal{O}(40n^3 + np^4)$. Calibration is not considered in the above analysis. If calibration is performed for n_c times, then the worst case complexity is $\mathcal{O}(n_c(40n^3 + np^4))$. In experimental evaluation, it is observed n_c is about 2 on average when the beam search width is set to 5. Therefore, the worst case complexity with calibration is $\mathcal{O}(80n^3 + 2np^4)$.

4 The Kernel Learning Interpretation of the Algorithm

From the viewpoint of kernel learning, it can be shown that the ℓ_1 regularized rule learning algorithm is equivalent to multiple kernel learning [10].

Multiple kernel learning (MKL) attempts to learn the optimal kernel matrix from data. Different kernels reflect different facets of the data set. MKL aims to combine these kernels in an optimal manner such that the obtained kernel matrix can best represent the original data set, as follows:

$$\min_{\boldsymbol{d}} J(\boldsymbol{d}) \quad s.t. : \sum_{m=1}^{M} d_m = 1, d_m \geq 0 \tag{10}$$

where $J(\boldsymbol{d})$ is the optimal value of the following problem:

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} -\frac{1}{2}\sum_{i,j}\alpha_i\alpha_j y_i y_j \sum_{m=1}^{M} d_m K_m(x_i, x_j) + \sum_i \alpha_i \\ &s.t. \sum_{i=1}^{n} \alpha_i y_i = 0, 0 \leq \alpha_i \leq t' \end{aligned} \tag{11}$$

Our ℓ_1 regularized algorithm can be proved to be equivalent to performing MKL with a linear kernel set induced by rules. First of all, the definition of linear kernel induced by rules is given as follows:

Definition 1. *Given a rule R_m, the linear kernel K_m induced by it is as follows:*

$$K_m(x_i, x_j) = < \tilde{\phi}_m(x_i), \tilde{\phi}_m(x_j) > \begin{cases} 1/\mathcal{N}_m^2, B \cup \{R_m\} \vDash x_i \cap B \cup \{R_m\} \vDash x_j \\ 0, \qquad otherwise \end{cases} \tag{12}$$

Theorem 5. *ℓ_1 SVM (7) is equivalent to the multiple kernel learning problem (11) with the linear kernel set $\{K_m\}_{m=1}^M$ constructed as in definition 1.*

Proof: Due to the block-sparsity as shown in [1], MKL (11) is equivalent to the following problem:

$$\min_{\beta} \sum_{i=1}^{n} L(y_i, \mathbb{B}^T \Phi(x_i)) \quad s.t. \|\mathbb{B}\|_{block} \leq t' \tag{13}$$

where L is the hinge loss, $\Phi = (\phi_1, ..., \phi_M)$, ϕ_m is the feature space induced by K_m, $\mathbb{B} = (\beta_1, ...\beta_M)$, β_m is the corresponding coefficient vector, $\|\mathbb{B}\|_{block} = \sum_{m=1}^{M} \|\beta_m\|_2$. According to definition 1, $\tilde{\phi}_m$ are single dimensional features, and thus (13) is simplified to (7). There is a one-to-one correspondence between t and t'. Consequently, (11) is equivalent to (7). □

The resulted kernel matrix is the optimal linear combination of the linear kernels induced by R_m, i.e., $K = \sum_{i=1}^{M} d_m^* K_m$. Our approach is actually a kernel matrix learning algorithm based on the obtained rules. Each rule provides a specific view, and these facets are finally integrated in the overall kernel matrix K. According to the kernel construction, the rules with lower complexity may have greater contribution to the final kernel matrix K. Therefore, as a kernel learning method, our approach is more flexible than the recent algorithms [7] which attempts to integrate ILP and kernel methods.

5 Experiments

As a general-purpose algorithm, our sparse rule combination approach is applicable to any rule generator such as FOIL, Progol, etc. In experiment, we realized two sparse rule combination algorithms respectively based on FOIL and Progol, denoted as ℓ_1FOIL and ℓ_1Progol. Both FOIL [15] and Progol [14] process data in a divide-and-conquer way, which consists of two iterative steps: the inner loop generates a single rule at each iteration, while the outer loop seeks for a set of rules to cover all positive observations. The difference lies in the way how a single rule is generated: FOIL can be considered as a first-order decision tree algorithm guided by FOIL-gain while Progol conducts a general-to-specific search in the theta-subsumption lattice of a single clause hypothesis through inverse entailment. The algorithms for comparison include **kFOIL** [11], **FOIL** [15], Rooted Kernel(**RKernel**,[19]), and Boosting FOIL (**BFOIL**). For **FOIL**, maximum number of clauses in a hypothesis was set to 25, maximum number of literals in a clause was set to 10, and a beam search with beam width 5 was performed instead of simple greedy search. For **BFOIL**, AdaBoost.M1 [5] was adopted for combining rules and the stepsize was fixed to 0.01 since some theoretical and experiment research [21,6] demonstrate this modification leads to better prediction accuracy, and the same parameters for FOIL were used. **kFOIL** is an algorithm that uses SVM as the score function for FOIL instead of the FOIL-gain. We used the same parameters as in **FOIL**. For the SVM part, a polynomial kernel of degree 2 was used. **RKernel** belongs to the family

of graph kernel [7] algorithms in which structural information of data based on graph theory is utilized to construct predefined kernels for SVM. The walk length and the discount factor for **RKernel** were selected from $\{1, 2, 4, 5, 10\}$ and $\{0.1, 0.5, 0.8, 1, 2, 10\}$ through cross validation. For ℓ_1FOIL and ℓ_1Progol, we also substituted simple greedy search with beam search with width 5, and there was no limitation on the number of clauses in a hypothesis and the number of literals in a clause because the complexity of the hypotheses would be controlled via ℓ_1 regularization. The LIBSVM implementation [2] was employed for the SVM part and the regularization constant C was selected from $\{0.1, 1, 10, 100, 1000, 10000\}$ through cross validation. For all of the algorithms, the validation technique was utilized to determine when to stop training. In each fold of cross validation (10-fold cross validation was performed for all data sets), the data set was split to 3 parts, 70% for training, 20% for validating and 10% for testing.

5.1 Datasets

For comparison, we evaluated the above algorithms on ten real world classification datasets, including Mutagenesis [18], Alzheimer [8], NCTRER [4] and PTC[1]. On Mutagenesis (230 observations, MUT) the problem is to predict the mutagenicity of a set of compounds. Alzheimer consists of four independent datasets, each of which corresponds to a desirable property of drugs against Alzheimer's disease: inhibit amine reuptake (686 observations, ALR), low toxicity (886 observations, ALT), high acetyl cholinesterase inhibition (1326 observations, ALC) and good reversal of memory deficiency (642 observations, ALM). NCTRER (232 observations,NCT) is extracted from the EPA's DSSTox NCTRER database, the problem is to predict estrogens' binding activity for the estrogen receptor. On all of the above datasets, we used the atom and bond information only. PTC contains 417 compounds, each of which is labeled based on whether it is carcinogenic to female rats (349 observations, PFR), female mice (351 observations, PFM), male rats (336 observations, PMR), and male mice (344 observations, PMM). Following [19] and others, we treat any molecule labeled "CE", "SE" and "P" as positive, "NE" and "N" as negative, ignore other unsure classifications.

5.2 Results

The 10-fold cross-validation result of the prediction accuracy is shown in Table 4. As demonstrated therein, ℓ_1FOIL significantly outperforms the other algorithms on six out of ten tasks. In addition, on none of the tasks ℓ_1FOIL and ℓ_1Progol performed significantly worse than the other algorithms. It is worth noting that ℓ_1FOIL and ℓ_1Progol have similar performance on nearly all of the tasks although their rule generation mechanisms are totally different. This indicates that as a meta-algorithm the performance of our approach is insensitive to the choice of different rule generators. This phenomenon may be due to the following three reasons: firstly, in relational learning the true model for a certain task often consists of several rules, each of which accounts for part of observations; secondly, FOIL and Progol are highly expressive in that they both have very large hypothesis spaces, and thus the rules for the true model is probably contained

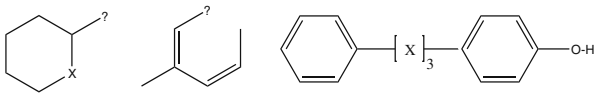
[1] http://www.predictive-toxicology.org/ptc/

Table 4. 10-fold cross validation predictive accuracy results. •/▲ indicate that the result for ℓ_1FOIL/ℓ_1Progol is significantly better than that of the corresponding method. On none of datasets, performance of ℓ_1FOIL or ℓ_1Progol is significantly worse (paired samples t-test, $\alpha = 0.05$). The second row for each dataset is the number of obtained rules on average.

Dataset	ℓ_1FOIL	ℓ_1Progol	kFOIL	FOIL	Rkernel	BFOIL
MUT	85.3 ± 2.8	86.9 ± 1.5	82.7 ± 10.3	75.3 ± 11.7 • ▲	85.4 ± 6.1	81.2 ± 7.4
	7.2 ± 2.1	8.4 ± 1.3	13.4 ± 5.2	7.5 ± 1.4	− − −	21.2 ± 2.2
ALR	92.6 ± 3.1	89.2 ± 4.5	87.9 ± 4.9•	76.5 ± 12.1 • ▲	83.1 ± 7.8 • ▲	87.3 ± 3.2•
	13.4 ± 4.8	15.7 ± 2.6	13.4 ± 1.9	12.7 ± 3.1	− − −	20.3 ± 3.7
ALT	91.6 ± 1.1	92.3 ± 2.4	89.5 ± 4.0	78.2 ± 5.2 • ▲	88.3 ± 8.1	86.3 ± 5.2
	12.1 ± 2.3	16.7 ± 3.6	22.3 ± 7.5	13.4 ± 4.1	− − −	16.9 ± 3.5
ALC	91.3 ± 2.4	88.2 ± 3.9	87.4 ± 3.8•	67.4 ± 5.5 • ▲	84.6 ± 6.8 • ▲	81.2 ± 7.8 • ▲
	8.4 ± 2.7	9.5 ± 3.6	15.6 ± 6.2	13.2 ± 3.4	− − −	17.1 ± 9.1
ALM	85.7 ± 4.2	89.3 ± 4.6	81.3 ± 3.5 • ▲	61.3 ± 12.8 • ▲	81.9 ± 3.6 • ▲	75.3 ± 2.1 • ▲
	11.3 ± 2.7	12.8 ± 5.4	7.9 ± 2.1	12.4 ± 5.3	− − −	21.4 ± 5.9
NTC	86.3 ± 2.3	85.2 ± 3.8	78.2 ± 8.7 • ▲	57.8 ± 6.3 • ▲	85.2 ± 7.2	79.2 ± 2.5 • ▲
	12.2 ± 2.2	13.1 ± 2.1	16.7 ± 7.5	14.3 ± 2.2	− − −	18.9 ± 4.1
PFR	75.2 ± 5.1	78.2 ± 3.4	82.2 ± 12.1•	57.9 ± 9.2 • ▲	67.3 ± 11.8•	73.3 ± 4.6▲
	17.3 ± 6.0	19.2 ± 2.6	23.5 ± 8.2	12.3 ± 4.9	− − −	13.9 ± 5.2
PFM	77.4 ± 7.1	78.6 ± 2.3	62.4 ± 7.2 • ▲	52.3 ± 10.1 • ▲	63.8 ± 6.1 • ▲	72.3 ± 6.4 • ▲
	18.7 ± 8.1	16.5 ± 3.2	21.3 ± 6.4	12.1 ± 3.1	− − −	19.4 ± 8.9
PMR	68.5 ± 3.4	66.5 ± 2.7	57.5 ± 5.2 • ▲	63.5 ± 6.4•	61.5 ± 7.3 • ▲	68.2 ± 8.4
	13.2 ± 5.5	17.2 ± 3.1	24.1 ± 2.1	13.6 ± 5.3	− − −	18.2 ± 7.2
PMM	70.9 ± 3.7	71.2 ± 5.2	72.1 ± 2.1	67.1 ± 9.2•	65.2 ± 8.9▲	63.8 ± 4.1•
	8.3 ± 2.1	11.6 ± 3.9	12.3 ± 2.4	7.9 ± 5.8	− − −	25.1 ± 3.2

in their hypothesis spaces; finally, the most relevant rules can be automatically picked out from the numerous candidates via ℓ_1 regularization. Therefore, the true model can be well approximated by the combination of expressive rule generators and ℓ_1 regularization. Besides the outstanding approximation ability, there are two more reasons for the success of our approach: firstly, due to ℓ_1 regularization the variance of the generalization performance is much less than the other algorithms; secondly, the entire solution path is generated and thus the risk of missing the right model is greatly reduced. Although boosting also performs rule combination, its performance is worse than our algorithms. The reason may be that due to the hinge loss our algorithm can pick out the support vectors and thus is less influenced by the noise than boosting. The comparison between **RKernel** and other algorithms is somewhat unfair in that there are two more tunable parameters for **RKernel**, i.e., the walk length and the discount factor. This partly explains its competitive performance in the evaluation. However, its interpretability is not as straightforward as that of our algorithms.

The 10-fold cross-validation result of the number of rules is also shown in Table 4. The number of rules generated by our algorithms is comparable to that by other algorithms. Although we claimed that sparse rule combination is achieved through our framework, the obtained rules is not necessarily fewer than that of other algorithms. The sparsity only implies that the rules involved in the true model may be chosen automatically. If for a certain task, the corresponding true model consists of many rules,



(a) Heteroatom: The atom at X is O, S or N. (b) Aromatic Ring (c) Phenol3nPhenyl: X can be any kind of atoms.

Fig. 2. Three Obtained Rules for NCTRER Dataset by ℓ_1FOIL

Table 5.

(a) The Influence of Beam Width on the Performance of ℓ_1 FOIL. A stands for prediction accuracy, R stands for the number of obtained rules and C stands for calibration times. The results are mean values of 10-fold cross validation.

Dataset	$Width = 1$			$Width = 3$			$Width = 5$		
	A	C	R	A	C	R	A	C	R
MUT	80.9	28.2	8.1	86.7	7.3	8.8	85.3	1.2	7.2
ALR	84.2	7.4	15.9	88.9	4.6	11.1	92.6	1.3	13.4
ALT	87.9	31.2	7.6	89.1	15.8	12.6	91.6	0	12.1
ALC	88.1	14.7	12.4	87.5	2.3	10.5	91.3	1.7	8.4
ALM	82.4	12.3	12.8	86.4	3.5	15.7	85.7	2.1	11.3
NCT	83.6	18.6	9.1	87.8	5.4	16.3	86.3	2.6	12.2
PFR	70.1	15.4	12.4	72.1	6.7	12.4	75.2	2.3	17.3
PFM	73.3	12.6	22.1	75.6	8.9	16.9	77.4	1.3	18.8
PMR	63.4	8.7	9.1	65.9	7.3	17.2	68.5	4.5	13.2
PMM	62.8	9.3	9.5	71.2	6.9	9.1	70.9	2.5	8.3

(b) The Influence of Rule Complexity Penalty on the Performance of ℓ_1 FOIL and ℓ_1 Progol. A stands for prediction accuracy and L stands for the number of literals in each rule. The results are mean values of 10-fold cross validation.

Dataset	$\mathcal{P}_a$FOIL		$\mathcal{P}_b$FOIL		$\mathcal{P}_a$Progol		$\mathcal{P}_b$Progol	
	A	L	A	L	A	L	A	L
MUT	85.3	2.4	83.1	3.3	86.9	2.6	85.4	5.1
ALR	92.6	4.7	93.2	6.4	89.2	3.2	91.3	4.7
ALT	91.6	3.6	92.7	7.5	92.3	4.1	89.7	4.9
ALC	91.3	3.2	88.9	4.8	88.2	3.2	91.2	5.6
ALM	85.7	5.4	80.4	3.5	89.3	3.7	90.9	4.8
NCT	86.3	4.2	82.1	4.6	85.2	2.3	86.6	3.2
PFR	75.2	2.6	73.5	4.5	78.9	3.2	82.3	5.8
PFM	77.4	3.8	76.9	5.4	78.6	5.7	77.1.9	4.3
PMR	68.3	3.4	72.2	3.6	66.5	4.8	70.2	5.3
PMM	70.3	2.3	68.1	4.3	71.2	3.1	68.3	4.1

then the performance of an algorithm apt to choose fewer rules cannot be expected to be satisfactory. For **RKernel**, the corresponding place is filled with "- - -" instead of a real number because it is based on kernel design rather than rule combination. For NCTRER dataset, three rules among the top ten rules (ordered by coefficients) are shown is Figure 2. This result is consistent with the study on NTCRER [4].

In Theorem 4, we proved that if at each iteration the optimal rule can be generated, then the obtained rule set is globally optimal. However, in practice only a suboptimal rule can be generated for each iteration. For beam search based rule generators, the approximity between the optimal rule and the suboptimal rule depends on the width of beam search. Therefore, the prediction accuracy, the calibration times (refer to section 3.3) and the number of rules can all be influenced by the beam width. We tried different search width (1, 3, 5) and the result is shown in Table 5(a). It can be observed that the calibration times decreases as the search width increases and this indicates that the chance of missing the optimal rule for a certain iteration decreases as the search width increases. The generalization performance improves as the search width increases as we expected. The number of obtained rules decreases as the search width increases. This implies that when the suboptimal rules chosen at each step are far from the optimal rules, more rules are needed to approximate the true model.

Another interesting question is how the size of rule set and prediction accuracy will be influenced by different designs of rule complexity penalties. For comparison , we

evaluated ℓ_1FOIL and ℓ_1Progol based on two different rule complexity penalties: ($\mathcal{P}_a$) C_m is defined as the number of the literals in R_m ; ($\mathcal{P}_b$) all C_m are just set to 1, i.e., no additional rule complexity penalty. The result is shown in Table 5(b). The prediction accuracy under the two kinds of penalties are close to each other while the literals in each rule are fewer for the case of literal number penalty.

6 Conclusion and Future Work

In this paper we bring out an ℓ_1 regularized rule learning and combining approach for relational learning, which formally introduces ℓ_1 regularization into the infinite space of horn clauses. Our approach is applicable to any horn clause generators such as FOIL, Progol, etc. Moreover, due to the piecewise linearity of hinge loss the entire solution path can be generated for a given task. It is proved that if a locally optimal rule is generated at each iteration, then each cross-section of the path is an optimal rule set given the ℓ_1 constraint on coefficients. From the kernel learning viewpoint, our approach is equivalent to multiple kernel learning and the information carried in the obtained rules is optimally fused into the overall kernel matrix. The proposed algorithm is competitive with the other algorithms for comparison. In future, the approach presented here will be extended to tackle the regression tasks.

Acknowledgments. We gratefully acknowledge Kuijun Ma for her insightful remarks. This work was partially supported by National Basic Research Program of China under Grant No.2004CB318103 and National Natural Science Foundation of China under award No.60835002.

References

1. Bach, F.R., Lanckriet, G., Jordan, M.I.: Multiple Kernel Learning,Conic Duality, and the SMO Algorithm. In: 21st international conference on Machine learning. ACM Press, New York (2004)
2. Chang, C.C., Lin, C.J.: LIBSVM: a library for support vector machines (2001), `http://www.csie.ntu.edu.tw/~cjlin/libsvm`
3. Dzeroski, S., Lavrac, N.: An Introduction to Inductive Logic Programming, Relational Data Mining. Springer-Verlag New York, Inc., USA (2001)
4. Fang, H., Tong, W., Shi, L.M., Blair, R., Perkins, R., Branham, W., Hass, B.S., Xie, Q., Dial, S.L., Moland, C.L., Sheehan, D.M.: Structure-activity relationships for a large diverse set of natural, synthetic, and environmental estrogens. Chem. Res. Tox. 14, 280–294 (2001)
5. 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)
6. Friedman, J.H.: Greedy function approximation: A gradient boosting machine. Annals of Statistics 29, 1189–1232 (2001)
7. Gaertner, T., Lloyd, J.W., Flach, P.A.: Kernels and Distances for Structured Data. Machine Learning 57, 205–232 (2004)
8. King, R.S., Sternberg, M.J.E., Srinivasan, A.: Relating Chemical Activity to Structure: An Examination of "ILP" Successes. New Generation Computing 13(3-4), 411–433 (1995)
9. Knight, K., Fu, W.J.: Asymptotics for lass-type estimators. The Annals of Statistics 28(5), 1356–1378 (2000)

10. Lanckriet., G., Cristianini, N., Bartlett, P., El Ghaoui, L., Jordan., M.I.: Learning the Kernel Matrix with Semi-horn Programming. EECS Department, University of California, Berkeley (2002), http://www.eecs.berkeley.edu/Pubs/TechRpts/2002/5765.html,UCB/CSD-02-1206
11. Landwehr, N., Passerni, A., De Raedt, L., Frasconi, P.: kFOIL: Learning Simple Relational Kernels. In: Proc. of the 21st Natl. Conf. on Artificial Intelligence, AAAI Press, Boston (2006)
12. Landwehr, N., Kersting, K., De Raedt, L.: nFOIL:Integrating Naive Bayes and FOIL. Journal of Machine Learning Research (JMLR) 8, 481–507 (2007)
13. Megiddo, N.: Progress in Mathematical Programming: Interior-Point and Related Methods. Springer, New York (1988)
14. Muggleton, S.: Inverse Entailment and Progol. New Generation Computing 13(3 and 4), 245–286 (1995)
15. Quinlan, J.R.: Learning Logical Definitions from Relations. Machine Learning 5(3), 239–266 (1990)
16. Rosset, S., Zhu, J.: Piecewise linear regularized solution paths. Annals of Statistics 35, 1012–1030 (2007)
17. Rosset, S., Swirszcz, G., Srebro, N., Zhu, J.: ℓ_1 Regularization in Infinite Dimensional Feature Spaces. In: Bshouty, N.H., Gentile, C. (eds.) COLT. LNCS (LNAI), vol. 4539, pp. 544–558. Springer, Heidelberg (2007)
18. Srinivasan, A., Muggleton, S., Sternberg, M.J.E., King, R.D.: Theories for Mutagenicity: A Study in First-Order and Feature-Based Induction. Artificial Intelligence 85(1-2), 277–299 (1996)
19. Wachman, G., Khardon, R.: Learning from Interpretations: A Rooted Kernel for Ordered Hyper- graphs. In: 24th international conference on Machine learning. ACM Press, New York (2007)
20. Zhang, T.: Statistical behavior and consistency of classification methods based on convex risk minimization. The Annals of Statistics 32, 56–85 (2004)
21. Zhang, T., Yu, B.: Boosting with early stopping: convergence and consistency. Annals of Statistics 33, 2005 (2003)
22. Zhao, P., Yu, B.: On Model Selection Consistency of Lasso. J. Machine Learning Research 7, 2541–2563 (2006)
23. Zhu, J.: Flexible statistical modeling. Ph.D. Thesis. Stanford University (2003)
24. Zhu, J., Rosset, S., Hastie, T., Tibshirani, R.: 1-norm support vector machine. In: NIPS, vol. 16, MIT Press, Cambridge (2003)