

# CREDOS: Classification using Ripple Down Structure (A Case for Rare Classes)

Mahesh V. Joshi\*

Vipin Kumar†

## Abstract

Ripple down rules (RDRs) are commonly used by the expert systems community because they make knowledge bases easy to use and efficient to maintain. We observe that RDRs offer a unique tree-based representation that generalizes the decision tree and disjunctive normal form (DNF) rule-based models, and specializes a generic form of the PNrul model. In this paper, we explore their use for learning predictive classifier models. Such models require to have a generalization capability, most commonly achieved with the help of pruning methods. Existing RDR induction algorithms are developed to build an initial knowledge base that will be used and modified by humans to explain every case correctly. They do not look at RDR as a predictive model, and hence offer very little measures against over-fitting. Existing pruning strategies developed by the data mining community cannot be directly used for pruning a RDR structure because of the uniqueness of the structure and the prediction process. In this paper, we propose a novel induction algorithm CREDOS. The key characteristic of CREDOS is its generic pruning framework. We provide a specific instantiation of it based on the minimum description length (MDL) principle. Using real-world datasets requiring prediction of rare classes, we compare CREDOS to other state-of-the-art algorithms. It exhibits significantly better or comparable performance, especially in predicting a wide variety of rarely occurring events.

## 1 Introduction and Motivation

Ripple-down-rules (RDRs) [7] were originally proposed for managing knowledge-bases, wherein they are used

for assigning a class to a given record. However, the goal behind the design of RDRs is not prediction per se. They were motivated by the desire to achieve ease of interpretation and efficiency in incrementally updating the knowledge-bases. In this paper, we intend to use RDR structure for a different goal: as a predictive model for the classification problem in machine learning; i.e., a model that correctly predicts the class of as many new cases as possible. In contrast with the original intended goal of always maintaining a consistent RDR structure by modifying it whenever a conflicting case is encountered, our goal translates into minimizing the need for such modifications. An obvious question to ask is why introduce yet another classifier model, when an abundance of them is already proposed in the literature [20]. The answer lies in the special kind of the classification problems we have been studying recently, and in the unique model structure offered by RDRs.

We have been recently looking at problems arising from important domains such as fraud detection, network intrusion detection, text categorization, and web mining. The problem is to learn effective classifier models for *rare* events, so as to predict large number of future occurrences of such events (i.e. high recall) with small false positive rate (i.e. high precision). In our study of rule-based models, we have found that the specialized models and algorithms such as PNrul [1, 15] perform significantly better than existing algorithms such as C4.5, C4.5rules [22], and RIPPER [6]. RDR structure fills an important gap in the spectrum of models offered by these algorithms. Briefly, RDRs can be introduced as a decision tree where each node has a predictive rule associated with it. The antecedent of this rule is a conjunction of conditions on attributes and the consequent is one of the classes. Apart from generalizing the decision tree model, it generalizes the rule-based model of RIPPER, and specializes a generalized version of the 2-phase PNrul model. This relationship is elaborated in Section 1.1.

Our experience with PNrul model indicates that the fundamental differences in how the models are induced and how predictions are made using them, lead

\*IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120, USA (joshim@us.ibm.com)

†Department of Computer Science, University of Minnesota, Minneapolis, MN 55455, USA (kumar@cs.umn.edu). This work was supported in part by the DOE/LLNL grant number W-7045-ENG-48 and by the NSF grant number IIS-0308264, the content of which does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred.

to the differences in their performance. Since RDRs offer a practical generalization of the two-phase PNrul model, we believe that they may be more effective in solving certain classes of rare class problems. In order to validate our belief, we propose a new algorithm, CREDOS<sup>1</sup>, for inducing RDRs. It has two stages: growth and pruning. Due to the nature of RDR structure, we offer multiple options in the growth phase with respect to the choice of class to be modeled at each node and ways of growing the rule within each node. We propose a generic one-pass pruning framework, and a specific version based on the Minimum Description Length (MDL). Although motivated by the decision tree pruning methods [22, 23, 18], our proposed framework is significantly different because of the fundamental differences in decision trees and RDRs.

Using various real-world and benchmark problems involving rare classes, we study various aspects of CREDOS. In particular, we compare various growth alternatives and demonstrate the efficacy of the pruning framework with respect to different types of problems. We then compare CREDOS with RIPPER, C4.5 tree, C4.5 rules, and PNrul, along two dimensions: functionality and ease of interpretation. The functional performance is measured in terms of recall and precision of the rare class, and is evaluated with and without the use of stratification wrapper, a common way of handling rare classes. Along this dimension, we observe that CREDOS learns comparable or significantly better models on a wide range of rare class problems. It is found to be especially preferable as an algorithm that can perform well under both unstratified and stratified class distributions. It tends to learn more interpretable models than most other methods, as measured by a metric of decision path length that we introduce to reflect the ease of explaining a prediction. However, we are not projecting CREDOS as the single best algorithm. When no stratification wrapper is used, the PNrul algorithm seems to be significantly better than CREDOS especially on relatively very tough problems involving relatively very rare classes. Only on such problems, one requires PNrul’s capability of having multiple rules at a node, which is missing from CREDOS. However, on problems other than these and whenever stratification helps, CREDOS tends to outperform PNrul.

#### Contributions of this work:

- Propose a new algorithm (CREDOS) for inducing the RDR structure with the goal of improving its generalization capability. A novel pruning strategy is proposed along with multiple alternatives of growing the structure.

<sup>1</sup>Classification using **Ripple Down Structure**

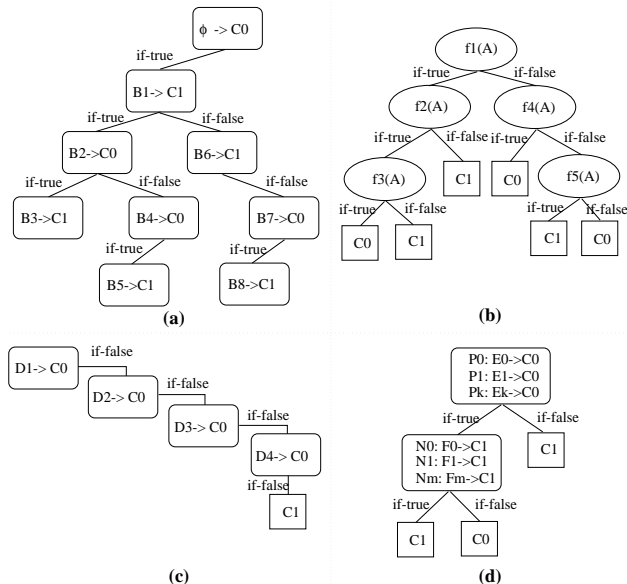


Figure 1: (a) Model based on Ripple-Down-Rule (RDR) structure. (b) Decision Tree model.  $f_i(A)$  is the decision based on a condition on univariate (C4.5) or multivariate (CART, OC1) function of attributes  $A$ . (c) Rule-based model. (d) Two-phase PNrul model.

- Empirical evaluation of CREDOS for the important problem of mining rare classes, to study its strengths and weaknesses with respect to other models.

#### 1.1 Spectrum of Tree- and Rule-based models

As mentioned in Section 1, one of the reasons for looking at RDRs is its unique model structure. Figure 1 illustrates its relationship with different classifier models used in data mining.

RDR (shown in Figure 1(a)) generalizes the decision-tree structure (Figure 1(b)) because it associates a DNF rule and a class to the decision of each *interior* node. While growing the RDR structure, a rule is induced for a specific class chosen according to some strategy, unlike the decision tree strategy of inducing a condition at a node that caters to all the classes at once. During prediction, if the rule at a node  $A$  of an RDR structure is satisfied and no rule in the subtree rooted at the if-true branch of  $A$  applies, then the decision is made at  $A$  predicting the class associated with  $A$ . This process can also be equivalently represented by a process of making predictions at the leaf. For every node  $B$  that has a null if-false branch, we can replace this branch by a leaf predicting the class associated with the node  $N$ , such that the path from  $B$  to root has its first if-true branch hanging from  $N$ . In our example of Figure 1(a), the null if-false branch of  $B4 \rightarrow C0$  node can be replaced

by a leaf predicting class  $C_1$ , which is the class associated with node  $B_1 \rightarrow C_1$ . After this is done for all the nodes, the classes associated with the interior nodes can be removed just keeping the rules. Note, however that this equivalence does not change the inherent nature of the induction phase, where a class must be associated with each interior node.

Furthermore, decision at each node of RDR is based on a conjunction of multiple univariate conditions. This is certainly a generalization over the single univariate condition used in C4.5 [22]. Other decision tree classifiers such as CART [4] and OC1 [21] allow multivariate conditions at each node based on the linear combinations of numerical attributes, which may be less interpretable than a conjunctive rule that can contain both numerical and categorical attributes.

If the if-true link of each node of an RDR structure is NULL, then it is simply a disjunction of the conjunctive rules of all the nodes. Thus, RDR structure also generalizes the DNF rule-set models (Figure 1(c)) that are induced by algorithms such as IREP [10] and RIPPER [6].

Note that RDRs have a single conjunctive rule at each node. If it is generalized to let each node have multiple disjoint rules predicting a given class with constraints on the coverage of the the target class, then the structure can be thought of as a generalized multi-phase version of the two-phase PNrul [1] model. The two-phase version, shown in Figure 1(d), has been used and studied in [1, 15]. Of course, a very general model, formed either by more than two phases of PNrul or by generalizing RDR to include multi-rule decisions at each node, will yield a much more complex classifier which may not remain very interpretable or might become too expensive to induce to justify its practical use.

In summary, we believe that the RDR structure with single-rule decisions, 2-phase PNrul, decision tree, and the DNF rule-list models, together offer a wide spectrum of *practical* rule-based models.

**1.2 Related Work** The research in RDRs has proceeded along the lines of generalizing the RDR structure [16, 2], but the primary motivating goal has always been efficient maintenance and ease-of-use of the knowledge bases. The existing algorithms for automatically inducing RDRs [11, 25, 17] from an existing knowledge base mainly cater to the traditional use of RDRs, and hence, they focus on growing the structure until it consistently satisfies most of the cases. However, in order to satisfy our classification goal, one must have a mechanism to avoid over-fitting of the induced RDR structure to the given training set. Two prominent RDR induction algorithms, Induct [11] and Cut95 [25], rec-

```

Given training set  $T$  consisting of records  $r = \{A_r, c_r, w_r\}$ .
( $A_r$ : attribute vector,  $c_r = \{C_0, C_1\}$ : class,  $w_r = \text{weight}$ )

LearnRDRmodel( $T$ )
   $H_1 = \text{InduceRDR}(T, \text{NULL}, -1)$ ; (Section 2.1)
  return  $H = \text{PruneRDR}(H_1)$ ; (Section 2.2)
end

InduceRDR( $S, D_h, c_p$ )
  if ( $|S| < \text{MinEx}$ ) return null node;
  if ( $S$  is impure)
     $C_h = \text{DecideClassToModel}(S, c_p)$ ;  $h_i = \text{index of } C_h$  (0 or 1)
    if ( $D_h$  is NULL)  $D_h = C_{(1-h_i)}$ 
     $R_h = \text{ChooseBestRule}(S, C_h)$ ; (Section 2.1.1)
     $S_t = \{r \in S : A_r \text{ satisfies } R_h\}$ .
     $p_h^R = \sum_{r \in S_t} w_r$  s.t.  $c_r = C_h$ 
     $n_h^R = \sum_{r \in S_t} w_r$  s.t.  $c_r = D_h$ 
     $h_t = \text{InduceRDR}(S_t, C_h, h_i)$ ;
     $h_f = \text{InduceRDR}(S - S_t, D_h, -1)$ ;
  else
     $C_h = \text{Larger class in } S$ ,  $D_h = \text{Smaller class in } S$ 
     $R_h = \text{NULL}$ ,  $p_h^R = n_h^R = 0$ ,  $h_t = h_f = \text{NULL}$ ;
  end
   $P_h = \sum_{r \in S} w_r$  s.t.  $c_r = C_h$ ,  $N_h = \sum_{r \in S} w_r$  s.t.  $c_r = D_h$ 
  return  $h = \{R_h, C_h, D_h, h_t, h_f, p_h^R, n_h^R, P_h, N_h\}$ 
end

DecideClassToModel( $S, c_p$ )
  if ( $c_p \neq -1$  AND AlternateTarget is true) return  $C_{(1-c_p)}$ .
   $n_i = \text{count of class } C_i \text{ in } S$ . ( $i = \{0, 1\}$ )
  if (ClassChoice is max-first) return  $C_i$  with larger  $n_i$ .
  if (ClassChoice is min-first) return  $C_i$  with smaller  $n_i$ .
end

```

Figure 2: CREDOS Algorithm: Inducing the RDR model.

ognize the need for pruning but only for individual rules, not for the structure as a whole. The Induct algorithm in [11] recognizes the need for pruning only to an extent of removing redundant clauses, It relates more to the goal of compressing the RDR structure, which has also been addressed in [24, 26] by removing repetitive RDR substructures, to make it more easy to use and maintain by humans. On the other hand, we prune the RDR structure to avoid over-fitting and improve its generalization prediction capability.

Cut95 [25] algorithm attempts to avoid over-fitting using a specific pre-pruning [9] strategy that stops refining a rule after it achieves an accuracy over certain threshold. However, efficacy of this parameter-driven method is not well established; they provide results on only two datasets. Our CREDOS algorithm's key feature is its systematic pruning framework that can perform pruning of individual rules as well as the structure as a whole, and we present empirical results on over 200 datasets.

## 2 CREDOS Algorithm

Given a training set  $T$  consisting of records each having multiple attributes, the classifier learning problem is to learn a predictive model for a given class attribute  $C$ ,

```

PredictClass( $H, r$ )
  { $C, s$ } = ClassifyRDR( $H, r$ );
  if ( $C$  is NULL)  $C = D_H, s = \text{score}(N_H - n_H^R, P_h - p_H^R)$ 
  return prediction of class  $C$  with score  $s$ 
end

ClassifyRDR( $h, r$ )
  if ( $R_h$  is NULL) return  $C_h$  and  $\text{score}(P_h, N_h)$ ; (Leaf Node)
  if ( $R_h$  covers  $r$ ; i.e.  $A_r$  satisfies  $R_h$ )
    if ( $h_t$  is not NULL) { $c, \text{score}$ } = ClassifyRDR( $h_t, r$ );
    else  $c = \text{NULL}$ 
    if ( $c$  is NULL) return  $C_h$  and  $\text{score}(p_h^R, n_h^R)$ 
    else return  $c$  and  $\text{score}$ 
  else
    if ( $h_f$  is not NULL) { $c, \text{score}$ } = ClassifyRDR( $h_f, r$ );
    else  $c = \text{NULL}, \text{score} = \text{to-be-filled-at-an-ancestor}$ 
    return  $c$  and  $\text{score}$ 
  end
end

```

Figure 3: Classification using CREDOS-induced RDR model.

such that the model correctly predicts most of the future occurrences of  $C$  with a small false positive rate. We focus on the binary classification problem for this paper; i.e., the problem involving only two classes. One way to extend it to multi-class problems is to learn multiple binary classifiers via the algorithm proposed here and combining the decisions of these classifiers [6, 1] using the probabilistic scores assigned to the decisions by the binary classifiers.

Figure 2 lists the main steps of our proposed CREDOS algorithm for inducing the RDR structure (Figure 1(a)) to solve the binary problem. Like many decision tree induction algorithms, CREDOS first grows an RDR structure to over-fit the training data, and then prunes it to improve the generalization capability (i.e. its performance on any unseen *test* set). The details of growth and pruning algorithms are however quite different from the tree induction algorithms.

The steps of the growth algorithm are given in the InduceRDR subroutine. It essentially generates a binary tree, where each node  $h$  is characterized by the rule  $R_h \rightarrow C_h$ , a default class  $D_h$ , and links to the two child subtrees,  $h_t$  and  $h_f$ . The  $h_t$  subtree corresponds to the if-true branch; i.e. it is accessed if  $R_h$  is satisfied by the record reaching  $h$ . The  $h_f$  subtree corresponds to the if-false branch. The statistics  $\{P_h, N_h, p_h^R, n_h^R\}$  are also kept at  $h$  to facilitate the pruning algorithm and assigning scores to decisions.  $P_h$  (resp.  $N_h$ ) is the sum of the weights on training set records of class  $C_h$  (resp.  $\bar{C}_h$ ), that reach  $h$ .  $p_h^R$  (resp.  $n_h^R$ ) is the sum of the weights on training set records of class  $C_h$  (resp.  $\bar{C}_h$ ), that reach  $h$  and satisfy  $R_h$ .

Before describing the details of InduceRDR algorithm and the pruning subroutine PruneRDR in Sections 2.1 and 2.2 respectively, we describe how a RDR

structure  $H$  is used for predicting the class of a new record,  $r$ . The prediction algorithm is given in Figure 3. The basic idea is that  $r$  is percolated down the tree based on whether a node's rule covers  $r$  or not. If  $r$  is covered by  $R_h$  and none of the rules in the subtree of  $h_t$  applies, then the prediction is  $C_h$ . Note that any non-null prediction made a node percolates back up the tree untouched, and is the final prediction. If  $r$  is not covered by  $R_h$  and none of the rules in the subtree of  $h_f$  applies, then the prediction job is passed onto the parent of  $h$ . At the root  $H$ , if none of the subtrees  $H_t$  and  $H_f$  have yielded non-NULL predictions, and if  $R_H$  does not cover  $r$ , then the default class  $D_H$  is predicted at the node. The score for this  $D_H$  prediction is computed using the statistics on the population where  $R_H$  does not apply. Currently, we simply use a Laplace correction version of the  $m$ -estimate [5] to implement the  $\text{score}(p, n)$  function (i.e.  $\text{score}(p, n) = (p + 1) / (p + n + 2)$ ). However, other sophisticated estimates of the probabilistic score can also be used [19, 8].

**2.1 Growth Phase (InduceRDR)** Like most decision tree and existing RDR induction algorithms, CREDOS grows the RDR structure in a recursive manner. The popular Induct [11] algorithm for inducing RDRs chooses the most frequent class in  $S$  to model at any given node. We instead offer four possible ways formed by the values of the user-specified parameters ClassChoice and AlternateTarget. As the algorithm in Figure 2 shows, one can either choose to model the most frequent or the least frequent class for the if-false branch of a node. One can do the same for the if-true branch, but here having an option of alternating the class between the parent and child also makes sense for the binary problem that we are focusing on, because we are in essence looking for an exception of the target class of the parent when the tree is built off the if-true branch.

Also note that the default class  $D_h$  associated with a node is not necessarily  $C_{(1-h_i)}$ , except at the root node. If this node were to have a null if-false branch,  $D_h$  essentially gives a quick way to predict the class of the records that come to this node but fail to be covered by  $R_h$ . This is because for all such records, the decision will be made at some ancestor of that node. The decision of that ancestor is percolated down to node  $h$  to form  $D_h$ . Storing  $D_h$  is necessary to achieve a one-pass pruning algorithm that is described later.

**2.1.1 Inducing Rule At A Node (ChooseBestRule)** The rule at each node is a conjunction of univariate conditions. The rule is induced greedily by searching the best condition to add from among all candidate conditions. For each numerical attribute  $A$ , all

distinct values  $v$  of  $A$  in  $S$  form the candidate conditions of the form  $A \geq v$  and  $A < v$ . For each categorical attribute  $B$ , all distinct values  $v$  of  $B$  in  $S$  form candidates  $B = v$  and  $B \neq v$ . As the conditions are added to  $R_h$ , the number of examples covered by it (i.e. its support) reduces, and its accuracy with respect to  $C_h$  is expected to improve. An evaluation metric such as information gain [6], gain ratio [22], or Z-number [1] can be used to select a condition that yields best combination of support and accuracy. Usually one can keep growing the rule until it covers no negative examples of  $C_h$ , but this often leads to the over-fitting of  $R_h$  to  $S$  (i.e.  $R_h$  may be significantly less accurate on any set other than  $S$ ). CREDOS offers a choice of using either of the two standard heuristics that attempt to avoid this, viz. pre-pruning and post-pruning, as elaborated in [9].

**2.2 Pruning Phase (PruneRDR)** We now propose a framework for pruning the RDR structure. Note that the growth phase prunes individual rules at a node. Here, we want to see whether some subtrees of the RDR structure can be removed to improve its generalization ability. The basic idea is to traverse the binary tree of the RDR structure starting from the leaf level going up the tree, use the statistics generated during the growth phase to evaluate different pruning options at each node, and to execute the option that achieves the best generalization capability  $G$ . Value of  $G$  can be assessed in various ways, such as the pessimistic error estimate used in C4.5 [22]. We choose an information theoretic approach based on the minimum description length (MDL) principle. It essentially tries to achieve a balance between the model complexity and model accuracy based on a hypothesis that low cost models tend to have higher generalization capability (Occam's razor [19]). The MDL principle is applied by computing the cost as the sum of the lengths of encoding a given model and the errors it makes on the given data. The model with the least such cost is chosen as the best model. This principle has been used in decision tree pruning [18, 23] as well as DNF rule-list induction [6]. We now describe how CREDOS uses it to prune the RDR structure.

Let us consider the pruning process at any node  $h$  of the binary RDR tree, assuming that the pruning has taken place up to the level of children of  $h$ . There are five options of pruning  $h$ , as illustrated in Figure 4, based on whether to keep one or both of the subtrees (options I- III), to remove both the subtrees and keep the rule at the node (option IV), or to entirely delete that node by removing its rule also (option V). Note that the option of removing the children while still

keeping its rule is unique to the RDR structure. It does not exist in the pruning algorithms of the traditional decision trees.

The information required to compute the cost of each option of pruning  $h$  is the statistics  $\{P_h, N_h, p_h^R, n_h^R\}$  stored at  $h$  by InduceRDR and the information that  $h$  receives from its children. In particular,  $h$  receives  $\{c_f, p_f, n_f\}$  from the if-false subtree ( $h_f$ ) and  $\{c_t, p_t, n_t\}$  from the if-true subtree ( $h_t$ ). An empty subtree sends  $\{0,0,0\}$ .  $c_f$  is the cost of keeping the  $h_f$  subtree.  $p_f$  (resp.  $n_f$ ) is the number of positive (resp. negative) examples of the target class of  $h_f$  for which at least one of the node-rules in the  $h_f$  subtree applies. Similarly for the if-true subtree. If the class associated with  $h_f$  or  $h_t$  is not the same as  $C_h$  (the class associated with  $h$ ), then the numbers  $p_f, n_f, p_t$ , and  $n_t$  are accordingly converted to correspond to the positive and negative examples of  $C_h$ .

The basic idea behind the cost computation is to compute the length of encoding the model given the data to which it is applied. Using Bayes rule, this can be computed as [19]

$$\text{cost}(\text{model}|\text{data}) = \text{cost}(\text{data}|\text{model}) + \text{cost}(\text{model}).$$

We now describe each of these two components.

**cost(data|model):** This is the cost of encoding the errors incurred due to the model. Let the set of examples reaching  $h$  be denoted by  $W$ . In order to facilitate error computation, we divide  $W$  into four disjoint subsets  $V, V_t, V_f$ , and  $U$ . The subset  $V$  contains examples whose class is decided at node  $h$  itself. Subset  $V_t$  (resp.  $V_f$ ) contains examples whose class is decided in the  $h_t$  (resp.  $h_f$ ) subtree. Finally, subset  $U$  has examples for which no decision can be made at any of the nodes in the subtree rooted at  $h$ . Let us also denote the set of examples covered by the rule  $R_h$  of  $h$  as  $V^R$ . Following relationships should be noted:

- (a)  $V^R = V \cup V_t$ ;
- (b) Sum of weights on examples of  $V_t$  (resp.  $V_f$ ) is equal to  $p_t + n_t$  (resp.  $p_f + n_f$ );
- (c)  $W = V \cup V_t \cup V_f \cup U = V^R \cup V_f \cup U$ .

The relationships (a)-(c) are invariant; thus, when an option removes either or both the subtrees, the containment of the sets changes. If the  $h_f$  subtree is removed (option II), then set  $V_f = \phi$ . So,  $V = V^R \setminus V_t$  is unchanged, but  $U = W \setminus V^R$ . If the  $h_t$  subtree is removed (option III), then set  $V_t = \phi$ . So,  $V = V^R$  and  $U = W \setminus V^R \setminus V_f$ . If both  $h_t$  and  $h_f$  are removed but  $R_h$  is kept (option IV), then both  $V = V^R$  and  $U = W \setminus V^R$  hold. If both  $h_t$  and  $h_f$  are removed and  $R_h$  is nullified (option V), then  $U = \phi$  and  $V = W$ .

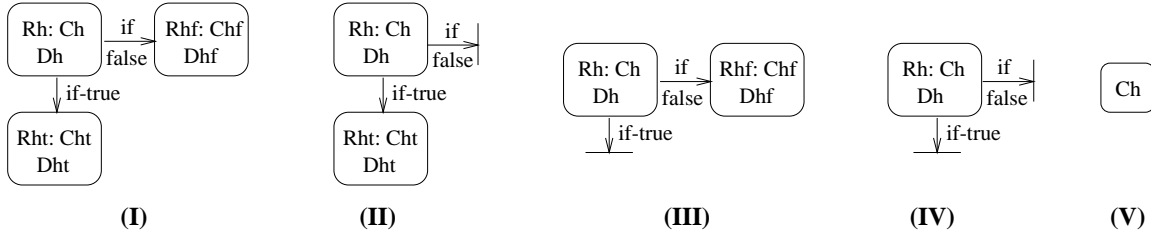


Figure 4: *Pruning Options for a node h: (I): Keep as-is, (II): Delete the if-false subtree, (III): Delete the if-true subtree, (IV): Delete both subtrees but keep the rule, (V): Delete both subtrees and the rule.*

Option	$N_V$	$e_V$	$N_U$	$e_U$	$c_M$	$c_s$
I	$Z_h^R - Z_t$	$n_h^R - n_t$	$Z_h - Z_h^R - Z_f$	$N_h - n_h^R - n_f$ , if $C_h = D_h$ $P_h - p_h^R - p_f$ , if $C_h \neq D_h$	$c^R + 2$	$c_t + c_f$
II	$Z_h^R - Z_t$	$n_h^R - n_t$	$Z_h - Z_h^R$	$N_h - n_h^R$ , if $C_h = D_h$ $P_h - p_h^R$ , if $C_h \neq D_h$	$c^R + 2$	$c_t$
III	$Z_h^R$	$n_h^R$	$Z_h - Z_h^R - Z_f$	$N_h - n_h^R - n_f$ , if $C_h = D_h$ $P_h - p_h^R - p_f$ , if $C_h \neq D_h$	$c^R + 2$	$c_f$
IV	$Z_h^R$	$n_h^R$	$Z_h - Z_h^R$	$N_h - n_h^R$ , if $C_h = D_h$ $P_h - p_h^R$ , if $C_h \neq D_h$	$c^R + 2$	0
V	$Z_h$	$\min(P_h, N_h)$	0	0	0	0

Table 1: *Components of Cost for various pruning options. ( $Z_h = P_h + N_h$ ,  $Z_h^R = p_h^R + n_h^R$ ,  $Z_f = p_f + n_f$ ,  $Z_t = p_t + n_t$ )*

In order to compute errors in  $V$  and  $U$ , we need to know the class that will be predicted for examples in each of these subsets. It is clear that  $C_h$  is predicted for  $V$ , when the rule  $R_h$  is kept. When  $R_h$  is nullified, the class for  $V$  is the majority class in  $W$ . Note that  $D_h$  was generated and percolated by InduceRDR so as to reflect the prediction for the examples that are not covered by any rule in the subtree of  $h$ . Thus, the class predicted for the subset  $U$  is  $D_h$ . For each of the pruning options, the total number of examples ( $N_V, N_U$ ) and the erroneously predicted examples ( $e_V, e_U$ ) in the subsets  $V$  and  $U$  respectively, are summarized in Table 1. The table shows two expressions for computing  $e_U$  because the positive ( $P_h, p_h^R, p_f$ ) and negative statistics ( $N_h, n_h^R, n_f$ ) are defined with respect to  $C_h$ , so if  $D_h \neq C_h$ , their roles need to be reversed for computing the errors. As the table indicates,  $N_V$  and  $e_V$  values are computed for only the predictions made at  $h$  and not at any of the subtrees of  $h$ .

The cost of encoding the subset containing examples with total weight of  $N$ , out of which the total weight on the erroneously predicted example is  $e$ , is given by:

$$(2.1) \quad L(N, e) = \log N - e \log(e/N) - (N - e) \log(1 - e/N)$$

This method of encoding a set containing two types of examples was defined in [23].

**cost(model):** This component includes the cost of encoding the rule  $R_h$  and the type of  $h$ . The node can be

of four types: no children, two children, only left child, and only right child. The type can thus be encoded using 2 bits. The cost of encoding rule  $R_h$  that is formed by a conjunction of  $l$  conditions out of all possible  $m$  conditions is computed using the Equation 2.2 as  $c^R = L(m, l)$ . This method of encoding a rule is also used by RIPPER [6]. The total value of  $\text{cost}(\text{model})$  is thus  $c_M = c^R + 2$ .

**Total cost [cost(model|data)]:** Using Table 1 and Equation 2.2, the total cost for an option is computed as  $c_h = L(N_V, e_V) + L(N_U, e_U) + c_M + c_s$ . The first two terms form the  $\text{cost}(\text{data}|\text{model})$  component by encoding errors occurring in the sets  $V$  and  $U$ ;  $c_M$  is the  $\text{cost}(\text{model})$  component for  $h$ ; and  $c_s$  is the cost of encoding the non-null subtrees of  $h$ . Note that  $c_s$  includes the costs of encoding the errors, rules, and nodes that occur in the subtrees. The option that yields the least value of  $c_h$  is chosen. After executing the actions of that option, only the  $\{c_h - L(N_U, e_U), p_{(W-U)}, n_{(W-U)}\}$  numbers are passed on to the parent of  $h$ , where  $p_{(W-U)} = P_h - N_U$  and  $n_{(W-U)} = N_h - e_U$ . This is because the set  $U$  will be newly computed at parent of  $h$ , and  $h$  should just supply the cost due to the predictions made in the subtree rooted at  $h$ . The application of MDL principle implies that the pruning option that achieves the smallest total cost at node  $h$  is chosen.

In summary, the pruning process starts at leaves, calculates the  $\text{cost}(\text{model}|\text{data})$  of each pruning option

at each node, applies MDL principle to chooses the best option, propagates the appropriate statistics to the parent, and continues in this manner till all the nodes are evaluated.

### 3 Experimental Results

In this section, we evaluate the pruning strategies of CREDOS and compare its effectiveness with other state-of-the-art algorithms along the three dimensions: functional performance, computational performance, and ease of interpretation. We focus on the rare class problems due to its emerging importance in various domains. As our results indicate, the differences in various methods become especially critical for rarer and tougher problems.

**3.1 Evaluation Strategy** We start with description of datasets used, evaluation criteria, and performance tuning methods used in our experiments.

**3.1.1 Datasets Used** We use real-world and benchmark datasets from two domains. The first set of problems is formed using various datasets at the UCI machine learning repository [3]. In particular we form binary problems for each class with a proportion of  $< 35\%$  present in the datasets: thyroid (allhyper, allhypo, anthyroid), annealing, chess (king-rook-king), LED, letter recognition, waveform, pima indians diabetes, Boston housing, Statlog (satimage, shuttle, vehicle, segment, DNA), contraceptive method choice (cmc). There are total 138 UCI problems formed.

The second set of binary real-world rare class problems is formed using the OHSU Medline document corpus of medical abstracts from years 1990 and 1991 [12]. Each document has multiple topics assigned to it such as Infant, Infection, Cell\_Line, etc. There are around 148,000 documents with 73,700 words remaining after stemming and stop-word removal. We randomly select 70 topics from those with at least 300 documents ( $> 0.2\%$  population). The topic with most records has around 5% population. For each selected topic, sets of top 75 distinguishing words are selected according to two different evaluation metrics. These sets are intersected to choose the final set of feature words for the binary problem of that topic. We formed 70 such problems.

**3.1.2 Performance Tuning** The parameters used for tuning the performance of different algorithms are described in Table 2. We keep aside a validation set on which the performance is measured to decide the best parameter combination.

**Note on use of stratification:** In the context of

rare classes, often some sampling method, also known as stratification, is employed to reduce the skew in the class distributions. Under-sampling randomly samples a given percentage of majority class examples from the original training set. Over-sampling increases the weight on each minority class example by a given factor. Various stratification factors used in our experiments are given in Table 2. An important point to note is that stratification does not always help a given algorithm improve the performance over that achieved with no stratification. In fact, because of the use of validation set for parameter tuning, it may so happen that stratification over-fits the data and results in a poorer performance than without stratification. Hence, we compare results with stratification of any two methods using only the problems for which the best performance obtained with stratification by each method is better than the performance obtained by that method without stratification.

**3.1.3 Evaluation Criterion** Two models A and B (generated by using different options of the same algorithm or by different algorithms) achieving  $(R_a, P_a, F_a)$  and  $(R_b, P_b, F_b)$  values of recall, precision, and  $F_1$ -measure respectively<sup>2</sup>, are compared by first comparing  $R_a$  with  $R_b$  and  $P_a$  with  $P_b$  using the p-test [27] at a 95% confidence-level. If  $R_a \gg R_b$  and  $P_a \ll P_b$ , or  $P_a \gg P_b$  and  $R_a \ll R_b$ , then A is the better model. If  $R_a \sim R_b$  and  $P_a \sim P_b$ , then A and B are comparable. Suppose  $R_a \ll R_b$  and  $P_a \gg P_b$ , or  $R_a \gg R_b$  and  $P_a \ll P_b$ . In such cases, if  $F_a - F_b > 1\%$ , then A is better; but if  $|F_a - F_b| \leq 1\%$ , then A and B are comparable. It follows that if A is not better than or comparable to B, then B is better. The suitability of this criterion is discussed in [14].

For a given problem  $P$ , multiple ( $> 2$ ) models are compared by first finding one best model using pairwise comparisons and breaking ties arbitrarily, and then comparing the best to others to determine the set of all winning models. For each winning model, we call  $P$  as the winning problem of the method by which the model was generated.

To compare two methods,  $M_i$  and  $M_j$ , over a set of problems, the sets of winning problems for each method are obtained, denoted by  $S_i$  and  $S_j$  respectively. Effectiveness of  $M_i$  versus  $M_j$  is judged by counting the number of problems in  $S_{ij} = S_i \cup S_j$  where  $M_i$  is better than, worse than, or comparable to  $M_j$ . We denote

<sup>2</sup>Given a set having  $n$  records of the rarer class  $C$ , if a classifier predicts total  $l$  cases to be of  $C$  out of which  $m$  are correct predictions, then the recall ( $R$ ) of the the classifier for  $C$  is defined as  $m/n$  and precision ( $P$ ) is defined as  $m/l$ .  $F_1$  of the classifier for  $C$  is then defined as  $2 \times R \times P / (R + P)$ .

Parameter	Algorithm(s)	Settings
Stratification	D,R,P,C	Oversampling {2.5, 5, 10}, Under-sampling {5%, 10%, 25%, 50%}
Recall Limits	P	{0.75,0.95} for P-phase, {0.0,0.7} for N-phase (refer to [15])
Number of Models	R,P	1 (only for rare class $C$ ), 2 (for both $C$ and $NC$ )
Rule evaluation Metric	D,R,P	Z-number [1], gain ratio [22], and information gain [6]
ClassChoice	D	min-first ( $\uparrow$ ), max-first ( $\downarrow$ )
AlternateTarget	D	true ( $A$ ); false, so decided by ClassChoice ( $C$ )
Rule pruning	D	Pre-pruning ( $E$ ); Post-pruning ( $T$ )

Table 2: Parameters Used for Different Algorithms. Notation:: D: CREDOS, R: RIPPER, P: PNrule, C: C4.5/C4.5rules.

	NoStrat		Strat	
	UCI	OHS	UCI	OHS
$ S_P $ - $ S_U $ - $ S_{eq} $	81-42-15	32-29-9	37-10-5	40-5-5

(a)

$F_1^{min}$ (%)	UCI (NoStrat)			
	$\rho$ (%)			
	(0.1, 4.2]	(4.2, 18.1]	(18.1, 35.5]	Any $\rho$
(40.7, 70.3]	4-4-1	4-17-1	5-9-1	13-30-3
(70.3, 86.1]	5-5-3	16-2-0	11-2-2	32-9-5
(85.1, 99.9]	18-2-4	4-0-2	14-1-1	36-3-7
Any $F_1$	27-11-8	24-19-3	30-12-4	81-42-15

$F_1^{min}$	OHSUMED (NoStrat)		
	$\rho \in (0.9, 2.8]\%$	$\rho \in (2.8, 19.6]\%$	Any $\rho$
(11.3, 32.6]%	1-11-2	2-7-1	3-18-3
(32.6, 55.3]%	4-5-2	6-5-2	10-10-4
(55.3, 92.0]%	9-1-0	10-0-2	19-1-2
Any $F_1$	14-17-4	18-12-5	32-29-9

(b)

Table 3: (a): Comparing Effect of the Pruning algorithm of CREDOS. (b): Pruning effect on various types of problems defined by  $\rho$  and  $F_1^{min}$ , when no stratification is used. Format:  $b$ - $w$ - $e$  implies that pruned model performs better than, worse than, or comparable to the un-pruned model on  $b$ ,  $w$ , and  $e$  problems, respectively.

these counts by  $b$ ,  $w$ , and  $e$ , respectively, and report the comparison in the  $b$ - $w$ - $e$  format.

Often, we also split the set of problems along two dimensions defined by  $\rho$  and  $F_1^{min}$ , where  $\rho$  is the proportion of the rare class ( $\rho$ ) in training set. Higher  $\rho$  implies low level of rarity and vice versa.  $F_1^{min}$  is the minimum  $F_1$  value attainable for a given problem by any winning method. If every method can achieve a high  $F_1$  value then the problem is not very difficult. However, if some model achieves a low  $F_1$  value, then it indicates that the choice of modeling method is crucial, and hence, we regard it as a tougher problem.

**3.2 Evaluating CREDOS Parameters** First, we demonstrate the effectiveness of CREDOS pruning strategy, and comment on the choice of growth options.

**3.2.1 Does Pruning Help CREDOS?** A given set of problems  $S$  is split into three subsets:  $S_P$ ,

$S_U$ , and  $S_{eq}$ .  $S_P$  is the set of problems for which pruning helps improve the performance.  $S_{eq}$  is the set of problems where the performance obtained with the pruned RDR structure is comparable to that of the un-pruned structure. Then  $S_U = S \setminus (S_P \cup S_{eq})$  is the set of problems where pruning worsens the performance. The effect of pruning is compared via the cardinalities of these subsets. Table 3(a) shows the performance using notations  $b = |S_P|$ ,  $w = |S_U|$ , and  $e = |S_{eq}|$ .

As the numbers indicate, pruning is clearly helping when stratification is used. Without stratification, pruning is helping more problems on UCI sets, but seems to have only marginal advantage on OHSUMED sets. A clearer picture is obtained by splitting up the NoStrat columns along  $\rho$  and  $F_1^{min}$  dimensions, as shown in Table 3(b). The effect of pruning is not as sensitive to the rarity ( $\rho$ ) as it is to how tough the problem is ( $F_1^{min}$ ). Only for very low values of  $F_1^{min}$ , pruning tends to produce over-simplistic models that perform poorly as compared to the un-pruned models. But, for most other problems, pruning helps to learn comparable or significantly better models, when no stratification is used.

### 3.2.2 Evaluating CREDOS Growth Options

There are eight ways of growing the RDR structure using CREDOS, based on the combinations of the values of ClassChoice, AlternateTarget, and Rule pruning parameters. We empirically compared all combinations for the two sets of problems to find the best combination, using the knock-out method<sup>3</sup>. Following guidelines emerged: Pre-pruning of rules at each node is better than post-pruning irrespective of the use of stratification. Without stratification, it is better to model the alternate target at the if-true child, and the larger class at the if-false child. With stratification, it is preferable to choose the smaller class to model at the if-true as well as if-false child nodes.

<sup>3</sup>Let comparison of combinations  $O_i$  with  $O_j$  yield  $b$ - $w$ - $e$  numbers (Section 3.1.3). If  $b > w$ , then choose  $O_i$ . If  $b < w$ , then choose  $O_j$ . If  $b = w$ , then choose the combination that wins more in such pairwise comparisons. The chosen option is compared to the next option.

### 3.3 Comparing CREDOS to Other Algorithms

We demonstrate effectiveness of CREDOS by comparing it with other classifier learning algorithms along two dimensions: functional performance and ease of interpretation. Depending on the domain of application, one or both of these criteria may be crucial.

**3.3.1 Functional Performance** For each problem, the evaluation criterion of Section 3.1.3 is used to choose the best model for each algorithm from among all the combinations of their respective parameter values listed in Table 2. Stratification factors are used only for results reported under Strat columns. As indicated in Section 3.1.2, only the problems that benefit from stratification are used to form the Strat column results. The best of C4.5 tree and C4.5rules is reported as C4.5. For CREDOS, if two sets of options have similar performance, the ties are broken in favor of the model with smaller model size (i.e. the total number of conditions in all the nodes).

The first set of results is shown on a small subset of UCI problems. This king-rook-king problem is regarded as one of the tougher ones [6]. We form 17 binary problems out of it, one for each of the classes. The comparative results are shown in Table 4. Observation of NoStrat (no stratification) columns indicates that CREDOS performance is either comparable ( $\sim$ ) or significantly better ( $\gg$ ) than the best competitor on most of the problems. Observation of the last column indicates that only a few problems benefit from stratification. This shows a need to report such results only for problems that indeed benefit from stratification, supporting our statement in Section 3.1.2. Looking at the Strat columns for only such problems, CREDOS seems to be utilizing stratification effectively more often than the other algorithms.

The set of 17 problems certainly indicates that capabilities of CREDOS are required for certain problems both with and without stratification. However, the set is too small to draw any conclusions. Hence, we now present cumulative comparative results on all the problems from the UCI and OHSUMED sets. They are shown in Table 5(a). One final observation of results in Table 4, such as those for classes *two* and *twelve* in either set of Strat and NoStrat columns, indicates that a rarer class problem (smaller  $\rho$ ) is not necessarily inherently more difficult than a problem involving relatively larger target class. This is the reason why we also decompose algorithm comparisons along the two dimensions of  $\rho$  and  $F_1^{min}$ , as was indicated in Section 3.1.3. Such decompositions of individual cells of Tables 5(a) are shown in Tables 5(b) and Tables 5(c) for UCI and OHSUMED sets, respectively.

First let us look at the results in NoStrat columns of Table 5(a). Note that the format of each entry is  $b-w-e$ ; i.e. the number 30-18-77 in CREDOS vs. RIPPER comparison in UCI column means that CREDOS is significantly better on 30 problems, significantly worse on 18 problems, and comparable on 77 problems. We can apply one-sided sign test [27] to the  $b$  and  $w$  numbers in such comparisons to find the overall significance of CREDOS being better than RIPPER. Using this test, CREDOS is significantly better than RIPPER, C4.5 tree, and C4.5rules with at least 99% confidence level on both UCI and OHSUMED problem sets, when no stratification is used. It is also significantly better than PNrul on the UCI problems at almost 95% confidence level. But, on the OHSUMED problems, CREDOS seems significantly worse when looking at cumulative results. A look at a corresponding decomposition of this entry along  $\rho$  and  $F_1^{min}$  dimensions in Table 5(c) indicates that PNrul is better than CREDOS on very tough or very rare problems (all four NoStrat cells, except the one corresponding to higher  $F_1^{min}$  and  $\rho$  ranges). CREDOS performance is comparable on moderately tough and moderately rare problems. In fact, notice that OHSUMED problems are, on an average, much rarer than UCI problems. So, looking at the decomposition of PNrul comparisons in both Tables 5(b) and 5(c) indicate that for relatively simpler problems (moderately tough or not very rare target class), CREDOS is comparable or significantly better than PNrul.

When stratification is used, PNrul performs significantly worse than CREDOS<sup>4</sup>, but RIPPER and C4.5/C4.5rules performance becomes comparable to CREDOS. An important point to note is that the results in Strat columns are reported only for those problems where each of the two methods being compared achieves better results with stratification factor  $\neq 1$ , than with factor = 1. For example, in the CREDOS vs. C4.5/C4.5rules comparison, only 41 (out of 138) UCI problems and 50 (out of 70) OHSUMED problems benefit from stratification. Breakdown of these results, shown in the right two columns of Tables 5(b) and 5(c), indicates that with stratified training sets, CREDOS has a slight edge over other algorithms when the problems are rarer or tougher. For not-so-rare and simpler problems, all the methods perform equally well, which supports our hypothesis that the algorithm choice matters primarily for rarer or tougher problems.

Overall, these results demonstrate following. CREDOS is preferable over C4.5rule/C4.5 tree and RIPPER, especially because of its consistency across differ-

<sup>4</sup> $b \gg w$  at  $> 95\%$  confidence-level as per the one-sided sign test.

DS ( $\rho$ )	NoStrat				Strat				Helps?
	C45	RIP	CRD	PNr	C45	RIP	CRD	PNr	
one (0.2%)	64.94 ~	63.53 ~	74.67 ~	59.02 ~	64.94 <<	80.00 >>	65.91 <<	59.02 <<	Yes
three (0.4%)	63.77 ~	59.74 ~	57.14 ~	49.23 ~	63.77 ~	59.74 ~	61.18 ~	45.10 <<	No
four (0.8%)	61.08 ~	52.70 ~	52.22 <<	59.00 <<	52.83 <<	52.70 <<	51.14 <<	59.65 >>	No
two (0.9%)	80.52 ~	81.60 ~	87.75 ~	80.92 ~	82.63 ~	81.60 ~	81.65 ~	79.72 ~	No
sixteen (1.4%)	69.21 ~	59.68 <<	65.31 ~	56.53 <<	69.21 ~	66.48 <<	60.42 <<	69.03 ~	No
five (1.5%)	65.41 ~	62.27 <<	71.76 ~	65.89 ~	74.07 >>	60.44 <<	71.76 <<	65.89 <<	Yes
seven (2.3%)	43.96 <<	23.58 <<	46.84 ~	46.43 ~	54.11 >>	46.36 <<	50.97 <<	46.43 <<	Yes
six (2.3%)	62.21 ~	47.66 <<	58.75 <<	62.50 ~	62.50 <<	60.30 <<	64.15 >>	62.50 <<	Yes
eight (5.2%)	55.97 <<	51.12 <<	55.63 ~	57.97 ~	56.80 ~	53.35 <<	54.91 <<	55.82 ~	No
nine (6.3%)	45.05 <<	29.86 <<	52.77 <<	54.44 >>	50.08 <<	49.11 <<	50.72 <<	54.44 >>	No
ten (7.4%)	42.05 <<	16.99 <<	46.77 >>	45.03 <<	42.05 <<	42.98 <<	48.68 >>	45.03 <<	Yes
fifteen (7.7%)	70.91 >>	62.46 <<	67.21 <<	66.04 <<	70.91 >>	64.30 <<	67.21 <<	65.22 <<	No
eleven (9.9%)	41.90 <<	8.02 <<	52.64 >>	50.92 <<	45.86 <<	47.53 <<	55.61 >>	50.92 <<	Yes
draw (10.2%)	93.44 <<	97.72 ~	97.00 ~	93.59 <<	97.37 ~	97.72 ~	97.89 ~	93.59 <<	No
twelve (12.9%)	51.21 <<	37.17 <<	54.64 ~	55.45 ~	55.81 ~	49.63 <<	55.67 ~	55.45 ~	No
thirteen (14.6%)	57.93 <<	47.54 <<	59.63 >>	58.51 <<	56.65 <<	53.00 <<	59.63 >>	58.51 <<	No
fourteen (16.0%)	68.36 ~	62.22 <<	69.03 ~	64.95 <<	69.92 ~	62.41 <<	69.03 ~	64.95 <<	No

Table 4: Comparative results for binary problems formed for all the classes of the king-rook-king problem in UCI repository. The original dataset was split into three sets: training (9446 total records), validation (4582), and test (14028). Validation set is used for tuning the parameters.  $F_1$  values are reported on the test dataset. NoStrat columns are results without stratification. Strat columns have results with best stratification factor from values listed in Table 2. Notation:  $\rho$ : proportion of the target class,  $\sim$ ,  $\ll$ ,  $\gg$  indicate equal, worse, and better performance than the best as per the comparison strategy described in Section 3.1.3. performance than the best, performance than the best. C45: best of C4.5 tree and C4.5rules, RIP: RIPPER, CRD: CREDOS, PNr: 2-phase PNrule.

CREDOS vs.	NoStrat		Strat	
	UCI	OHSUMED	UCI	OHSUMED
PNrule	32-19-75	9-31-30	22-6-12	13-8-28
RIPPER	30-18-77	16-0-23	12-7-22	3-6-38
C4.5/C4.5rules	36-19-71	15-1-24	15-7-19	9-5-32

(a)

UCI				
$F_1^{min}$	$\rho \in (0.1, 10.6]\%$	$\rho \in (10.6, 35.5]\%$	$\rho \in (0.1, 10.6]\%$	$\rho \in (10.6, 35.5]\%$
<b>vs. PNrule</b>	NoStrat		Strat	
(41.1, 73.9)%	8-6-16	8-4-11	6-2-2	6-1-5
(73.9, 99.9)%	11-4-26	5-5-22	5-2-2	5-1-3
<b>vs. RIPPER</b>	NoStrat		Strat	
(41.1, 73.9)%	9-2-15	11-1-8	5-0-3	3-1-8
(73.9, 99.9)%	5-8-32	5-7-22	4-4-3	0-2-8
<b>vs. C4.5</b>	NoStrat		Strat	
(41.1, 73.9)%	7-8-17	9-3-10	4-2-4	5-0-6
(73.9, 99.9)%	11-6-26	9-2-18	2-3-5	4-2-4

(b)

OHSUMED				
$F_1^{min}$	$\rho \in (0.1, 2.80]\%$	$\rho \in (2.80, 19.6]\%$	$\rho \in (0.1, 2.80]\%$	$\rho \in (2.80, 19.6]\%$
<b>vs. PNrule</b>	NoStrat		Strat	
(9.1, 40.4)%	2-10-4	0-8-4	3-4-8	4-1-5
(40.4, 90.7)%	2-6-11	5-7-11	3-0-6	3-3-9
<b>vs. RIPPER</b>	NoStrat		Strat	
(9.1, 40.4)%	5-0-1	3-0-1	2-3-9	0-1-9
(40.4, 90.7)%	4-0-9	4-0-12	0-1-9	1-1-11
<b>vs. C4.5</b>	NoStrat		Strat	
(9.1, 40.4)%	2-0-4	2-0-2	2-2-9	3-0-6
(40.4, 90.7)%	5-0-8	6-1-10	2-0-7	2-3-10

(c)

Table 5: (a): Comparing CREDOS performance with other algorithms with No stratification (NoStrat) and stratification (Strat). Comparison is made based on only the winning problems of each algorithm. Format: for the row of algorithm A, b-w-e implies that the CREDOS is better than, worse than, or comparable to A on b, w, and e problems, respectively. (b): Breaking down the comparison on UCI datasets over  $\rho$ - $F_1^{min}$  space. (c): Breaking down the comparison on OHSUMED datasets over  $\rho$ - $F_1^{min}$  space.

ent stratification strategies (i.e. class distributions) and learning difficulties. PNrul’s feature of having multiple rules at a single decision node, which is missing from CREDOS, is certainly required on tougher and rarer problems with skewed class distributions. But, for relatively simpler problems or problems not involving very rare classes (i.e. on stratified sets), PNrul does not perform as well as CREDOS. Note that we have obtained a stronger confirmation of this observation on some specially designed synthetic datasets as well [13].

**3.3.2 Ease of Interpretation** For the purpose of comparing ease of interpretation, the best overall model is chosen for each algorithm irrespective of whether it is obtained via stratification or not. We do pair-wise comparisons of CREDOS with other algorithms. We split the winning problems of C4.5 method into those obtained by the tree and rules versions.

We define the ease of interpretation of a model via its *decision-path length* (DPL). It is the maximum number of conditions that will be used to explain any decision it makes. For the tree based models (CREDOS and C4.5), DPL is the longest path from root to leaf measured in terms of sum of the number of conditions in the nodes on this path. For rule-list models (RIPPER and PNrul), DPL is the total number of conditions present in the model, because in the worst case the decision cannot be made until the last rule is applied, and decision made using the last rule has to be explained with the negation of all the previous rules. The comparisons of ease of interpretation are shown as scatter-plots in Figure 5. Each point  $(x,y)$  represents a problem, where  $x = \log(DPL_{CREDOS})$  and  $y = \log(DPL_{other})$ . For the problems falling above (resp. below) the  $y = x$  line, CREDOS learns a more (resp. less) interpretable model. The plots indicate that despite offering a generalization over the rule-list models, CREDOS is learning more interpretable models when compared to C4.5rules, RIPPER, and PNrul. Only C4.5 tree is more interpretable than CREDOS, but this is expected because C4.5 tree involves only single-attribute conditions vs. the DNF rules learned at every node of the RDR structure.

#### 4 Concluding Remarks

We believe that the ripple-down structure used widely by the expert-systems community offers a useful model that generalizes the traditional rule-based and tree-based models used by the data mining community. We explore its potential to learn predictive models with better generalization ability, by proposing a novel algorithm (CREDOS). The key distinguishing feature of CREDOS is its structure-pruning framework.

Our comprehensive experiments indicate that CREDOS effectively utilizes the unique nature of the RDR structure to learn comparable or better models for a variety of rare classes. CREDOS performs well under both unstratified and stratified class distributions. According to the decision path length metric, it tends to learn models that are often equally or more interpretable. Only the tough problems involving very rare classes require PNrul’s capabilities; but, CREDOS is preferable on most other problems. We believe that CREDOS and PNrul together offer two strong practical algorithms of learning effective rare class models.

#### References

- [1] R. C. Agarwal and M. V. Joshi. PNrul: A new framework for learning classifier models in data mining (A case-study in network intrusion detection). In *Proceedings of First SIAM Conference on Data Mining (SDM'01)*, Chicago, April 2001. Expanded version available as IBM Research Division Report RC 21719, April 2000.
- [2] G. Beydoun and A. G. Hoffmann. NRDR for the acquisition of search knowledge. In *Australian Joint Conference on Artificial Intelligence*, pages 177–186, 1997.
- [3] C. Blake and C. Merz. UCI repository of machine learning databases, 1998. <http://www.ics.uci.edu/~mllearn/MLRepository.html>.
- [4] L. Breiman, J. H. Friedman, R. A. Olshen, and C. J. Stone. *Classification and Regression Trees*. Wadsworth Inc., 1984.
- [5] B. Cestnik. Estimating probabilities : A crucial task in machine learning. In *Proc. of the 9th European Conference on Artificial Intelligence (ECAI-90)*, pages 147–149. Pitman, London, 1990.
- [6] W. W. Cohen. Fast effective rule induction. In *Proc. of Twelfth International Conference on Machine Learning*, Lake Tahoe, California, 1995.
- [7] P. Compton and R. Jansen. A philosophical basis for knowledge aquisition. *Knowledge Acquisition*, 2:241–257, 1990.
- [8] P. Domingos. Process-oriented estimation of generalization error. In *Proceedings of Intl. Joint Conf. on Artificial Intelligence (IJCAI)*, pages 714–721, 1999.
- [9] J. Fürnkranz. Pruning algorithms for rule learning. *Machine Learning*, 27(2):139–171, 1997.
- [10] J. Fürnkranz and G. Widmer. Incremental reduced error pruning. In *Proc. of Eleventh International Conference on Machine Learning (ICML-94)*, pages 70–77, New Brunswick, NJ, 1994.
- [11] B. R. Gaines and P. Compton. Induction of ripple-down rules applied to modeling large databases. *Journal of Intelligent Information Systems*, 5(3):211–228, 1995.
- [12] W. Hersh, C. Buckley, T. Leone, and D. Hickam. OHSUMED: An interactive retrieval evaluation and

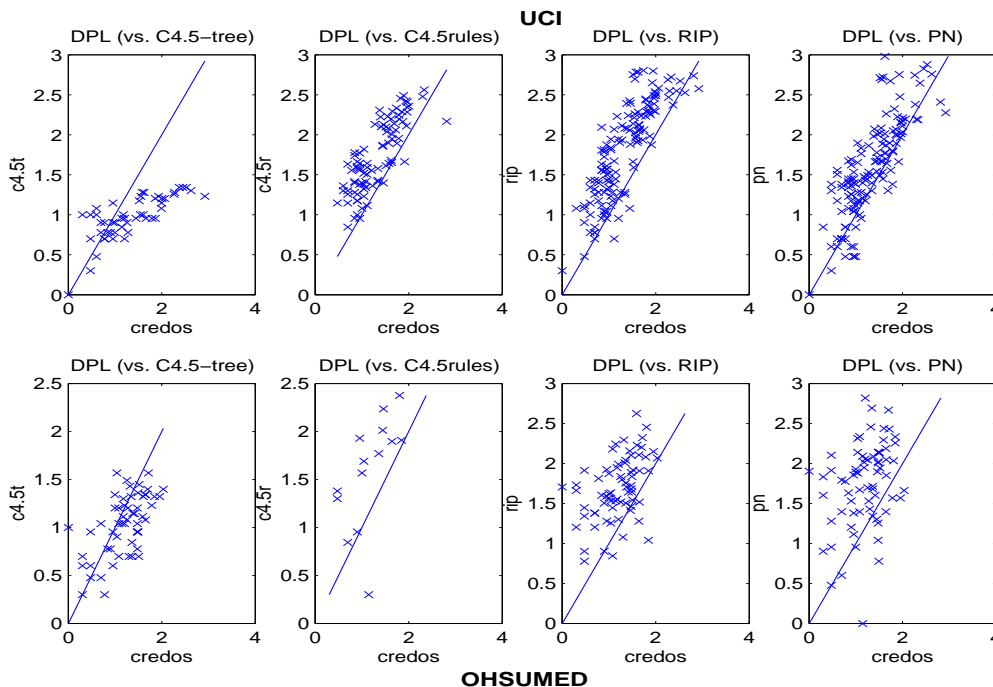


Figure 5: Comparing the ease of interpretation (DPL) of CREDOS with *C4.5-tree*, *C4.5rules*, *RIPPER*, and *PNrule* over the UCI (top row) and the OHSUMED (bottom row) datasets. A point  $\{x,y\} = \{\log(DPL_{CREDOS}), \log(DPL_{other\_algorithm})\}$ .

- new large test collection for research. In *Proc. of the 17th ACM SIGIR Conference*, pages 192–201, 1994.
- [13] M. V. Joshi. *Learning Classifier Models to Predict Rare Phenomena*. PhD thesis, University of Minnesota, 2002.
- [14] M. V. Joshi. On evaluating performance of classifiers for rare classes. In *Proc. of the Second IEEE International Conference on Data Mining (ICDM'02)*, Maebashi City, Japan, December 2002.
- [15] M. V. Joshi, R. C. Agarwal, and V. Kumar. Mining needles in a haystack: Classifying rare classes via two-phase rule induction. In *Proc. of ACM SIGMOD Conference on Management of Data (SIGMOD'01)*, pages 91–102, 2001.
- [16] B. Kang, P. Compton, and P. Preston. Multiple classification ripple down rules: Evaluation and possibilities. In *Proc. of 9th Banff Knowledge Acquisition for Knowledge Based Systems Workshop*, 1995.
- [17] J. Kivinen, H. Mannila, and E. Ukkonen. Learning rules with local exceptions. In *EUROCOLT: European Conference on Computational Learning Theory*, 1993.
- [18] M. Mehta, J. Rissanen, and R. Agrawal. MDL-based decision tree pruning. In *Proc. of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 216–221, 1995.
- [19] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.
- [20] S. K. Murthy. *On Growing Better Decision Trees from Data*. PhD thesis, Johns Hopkins University, 1995.
- [21] S. K. Murthy, S. Kasif, and S. Salzberg. A system for induction of oblique decision trees. *Journal of Artificial Intelligence Research*, 2:1–32, 1994.
- [22] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
- [23] J. R. Quinlan. Mdl and categorical theories (continued). In *Machine Learning: Proc. of the Twelfth Intl. Conference on Machine Learning*, Lake Tahoe, CA, 1995. Morgan Kaufmann.
- [24] D. Richards, V. Chellen, and P. Compton. The reuse of ripple down rule knowledge bases: Using machine learning to remove repetition. In *Proc. of Pacific Knowledge Acquisition Workshop (PKAW)*, Coogee, Australia, 1996.
- [25] T. Scheffer. Algebraic foundations and improved methods of induction of ripple down rules. In *Proc. of the 2nd Pacific Rim Knowledge Acquisition Workshop*, 1996.
- [26] H. Suryanto, D. Richards, and P. Compton. The automatic compression of multiple classification ripple down rule knowledge based systems: Preliminary experiments. In *Proc. of Third Intl. Conf. on Knowledge-based Intelligence Information Engineering Systems*, pages 203–206, Adelaide, Australia, 1999.
- [27] Y. Yang and X. Liu. A re-examination of text categorization methods. In *22nd Annual International Conference on Information Retrieval (SIGIR'99)*, pages 42–49, Berkley, August 1999.