Mining Needles in a Haystack: Classifying Rare Classes via Two-Phase Rule Induction

Mahesh V. Joshi*
IBM T. J. Watson Research Center and
University of Minnesota, Minneapolis
Department of Computer Science
joshim@us.ibm.com

Ramesh C. Agarwal
IBM T. J. Watson Research Center
P.O.Box 704
Yorktown Heights, NY 10598
ragarwal@us.ibm.com

Vipin Kumar
University of Minnesota
Department of Computer Science
Minneapolis, MN 55455
kumar@cs.umn.edu

ABSTRACT

Learning models to classify rarely occurring target classes is an important problem with applications in network intrusion detection, fraud detection, or deviation detection in general. In this paper, we analyze our previously proposed two-phase rule induction method in the context of learning complete and precise signatures of rare classes. The key feature of our method is that it separately conquers the objectives of achieving high recall and high precision for the given target class. The first phase of the method aims for high recall by inducing rules with high support and a reasonable level of accuracy. The second phase then tries to improve the precision by learning rules to remove false positives in the collection of the records covered by the first phase rules. Existing sequential covering techniques try to achieve high precision for each individual disjunct learned. In this paper, we claim that such approach is inadequate for rare classes, because of two problems: splintered false positives and error-prone small disjuncts. Motivated by the strengths of our two-phase design, we design various synthetic data models to identify and analyze the situations in which two state-of-the-art methods, RIPPER and C4.5rules, either fail to learn a model or learn a very poor model. In all these situations, our two-phase approach learns a model with significantly better recall and precision levels. We also present a comparison of the three methods on a challenging real-life network intrusion detection dataset. Our method is significantly better or comparable to the best competitor in terms of achieving better balance between recall and precision.

1. INTRODUCTION AND MOTIVATION

One of the important problems in data mining, commonly known as deviation detection, is that of modeling rarely occurring phenomena in the data. There are many situations in deviation detection, wherein rare deviant events have occurred in the past and have been identified, leading to a *labeled* data. For example, in network intrusion detection, consider attacks of type remote-to-local (r2l), on a computer that are made remotely by guessing a password or opening a ftp data connection. Provided that various data about the network activity are being collected, then the successful incidents of this attack would be rare yet they can be identified and later analyzed. The postulate behind the work presented and proposed in this paper is that rare events have their own unique signatures which can be learned from the given data. The focus is on the supervised learning of such signatures.

Classification is a powerful supervised technique in data mining that learns models from labeled data. Input to the classification problem is a set of observations from the real world that are recorded as a set of records, each characterized by multiple attributes. Associated with each record is a categorical attribute called *class*. Given a *training set* of records with known class labels, the problem is to learn a model for the class in terms of other attributes. The goal is to use this model to predict the class of any given set of records, such that certain objective function based on the predicted and actual classes is optimized.

We concentrate on general-to-specific rule-induction techniques of building classifier, because they are easily interpretable by humans, computationally tractable in most practical scenarios, and they have exhibited competitive performance in many application domains. General-to-specific techniques start building a DNF model with the most general rule, an empty rule, and progressively add specific conditions to it. The commonly used technique of sequential covering iteratively discovers multiple rules each having very high accuracy with respect to the target class that it predicts. Every time a rule is learned, the examples supported by it are removed before the next iteration. Each such discovered rule is expected to cover a disjoint signature of the target class. If the signature of the target class is pure, such that a rule corresponding to it covers very few negative examples; then this approach works fine.

We now describe two possible scenarios in which this se-

*Contact Author

ACM SIGMOD 2001 May 21-24, Santa Barbara, California, USA

quential covering approach of discovering high accuracy rules may fail. One is when the target class signature is composed of two components, presence of the target class and absence of the non-target-class, and the later component is not correctly or completely learned. This can happen, especially for rare classes, when a signature for the presence of the class is inherently impure by itself. As an example from intrusion detection, for a rare attack type r2l, a signature of the presence might be connection_type=ftp. However, this will also cover ftp connections made to flood the computer in a denial-of-service (dos) attack. Rule for r2l has to be refined by detecting signatures for the absence of dos attacks. In most existing techniques, tight accuracy constraints cause each rule to be refined immediately by adding more conjunctive conditions that detect the second component. However, each rule by itself may not cover a sufficient number of negative examples needed to learn correct signatures. We refer to this as the *problem of splintered false positives.*

The other problem that the existing sequential covering methods may face is that of small disjuncts [8, 15, 6], in which rules that cover small number of target class examples are more prone to generalization error than rules covering larger number of such examples. For prevalent classes, such rules usually arise in the later iterations of the algorithm, because the remainder data-set consists of small number of target class examples to learn from. However, we believe that for rare classes, such rules can start appearing very early, because the total number of target class examples is small to start with. We believe that the problem of small disjuncts manifests greatly in existing techniques, because of their tight constraint on the accuracy of each rule. This constraint translates into a low overall support for the rule[1]. From statistical point of view, decisions made with small evidential support are often unreliable.

Critical observation of these two problems leads us to the focal point of this paper, which is to analyze the effectiveness of our recently proposed two-phase rule-induction approach [1], especially in the context of building models for rare classes. According to our assessment, existing techniques may be inadequate because they try to achieve the objectives of high recall and high precision simultaneously, which may work well for prevalent classes with prevalent signatures. Our two-phase approach, called PNrule, separately conquers these two objectives in two phases. Like existing techniques, the first phase of our technique starts off with tight accuracy constraints. Unlike them, if a high accuracy rule cannot be found without sacrificing its support, then we favor a rule that has higher support but lower accuracy. Thus, in effect, our technique seeks high recall objective in first phase. We refer to the rules discovered in this phase as P-rules, as they detect presence of the target class. In the second phase, we seek precision. The key point is that before starting the second phase, we collect all the examples that are covered by P-rules, positive as well as negative, and learn rules to remove false positives from this collection. This can be contrasted to the way existing techniques refine each rule based on what it covers individually. We call the rules discovered in this second phase as N-rules, as they detect absence of the target class.

We hypothesize that there are two strengths of our two-phase method, especially for rare classes: *a.* the presence of second phase allows the first phase of PNrule to be less sensitive to the problem of small disjuncts, and *b.* the second phase of PNrule has better ability to learn signatures for the absence of target-class because it combines all the false positives together. In this paper, we experimentally validate these hypotheses by designing a variety of synthetic datasets and comparing our technique's ability to model rare classes in these datasets to that of two other existing state-of-the-art *core* methods of rule-induction, RIPPER [5] and C4.5rules [11][2]. We identify and analyze the situations in which existing methods of rule-induction fail, whereas our method yields a significantly better performance. In this paper, our focus is on the binary classification problem, where the goal is to learn signatures that distinguish given target-class from the rest[3]. The traditional evaluation metric of accuracy is not adequate when the target-class is rare. If the class is very rare, say 0.5%, then predicting everything to be of non-target-class can also achieve very high accuracy level of 99.5%. Hence, we believe that for rare classes, the classifier should be evaluated based on how it performs on both recall and precision. We use one such metric, called F-measure, widely used by the information retrieval community [14]. We start our comparative study with a simple model for generating synthetic datasets, and then progressively make the model complex. We observe the superiority of our method on all the models. We also present results that indicate that our technique performs significantly better for rare classes. As the proportion of target-class increases in the training set, performance of other methods catches up with that of our method.

We have made some improvements in the two-phase rule-inductions algorithm proposed in [1]. We illustrate the effectiveness of these improvements using a real-life dataset, which also further validates the better suitability of two-phase approach for rare classes. We use a dataset from the domain of network intrusion detection, which was supplied as part of the KDDCUP'99 classifier learning contest [7]. Unique features of this data-set make this an interesting application. We present results for two rare classes from this dataset. Not only is the improved method of this paper is better suited to achieve higher performance levels than its previous version, but it also significantly outperforms RIPPER and C4.5rules on this data-set.

The rest of the paper is organized as follows. We start with an overview of the key features of the improved two-phase algorith in section 2. Then, in section 3, various synthetic dataset models are designed, and comparative experimental results are presented and analyzed. In section 4, we present results on a real-life dataset. Finally, section 5 concludes the paper with summarizing remarks.

1.1 Related Work

Various rule-based classification algorithms have been proposed in the literature so far such as CN2 [3], the family of AQ algorithms [9], RIPPER [5], C4.5rues [11], and oth-

[1] Support is the total number of examples a rule covers, positive as well as negative

[2] We refer to these methods and our method as core methods because recently a few rule-induction methods are proposed (LRI [16], SLIPPER [4]), which use the concepts of boosting or bagging in the learning process [2, 12]. We believe that these methods are meta-techniques, and our two-phase induction can be used at the core of such techniques, just the way RIPPER is used at the core of SLIPPER.

[3] the framework's applicability to the multi-class problem with different costs of misclassification, was illustrated in [1].

ers [10]. We give overview of RIPPER and C4.5rules and explain why they face the problems of splintered false positives and small disjuncts.

We believe that the problem of small disjuncts is caused by the low, statistically insignificant support of the small coverage rules. Having small coverage can not be avoided for rare subclasses in a sequential-covering technique. However, what causes small coverage disjuncts to have small support is the relatively tight accuracy constraints in existing algorithms. Our technique relaxes accuracy constraints in its first phase, if required.

RIPPER and C4.5rules try to avoid overfitting of each disjunct, small or large, by pruning the rules. Existing pruning procedures in C4.5rules [11] or in RIPPER [5] work by first learning a set of most specific rules, and then generalizing each individual rule by removing some conjunctive conditions in it. In each iteration, RIPPER splits the remaining data-set into two random similar parts. One is used to grow the most specific rule and the other is used to generalize this rule immediately. As the remainder dataset size reduces, this approach may face two problems. First, the amount of training data reduces, so support of the rule decreases even further. Second, the estimates of error obtained from the small pruning set may not be reliable. Also, RIPPER uses principle of minimizing the description length (MDL) to avoid small disjuncts. However, from our experience and looking at the formula for computing MDL [11], small disjuncts tend to have longer lengths because of their small support. Hence, they have a higher chance of getting removed from the final rule-set, thus possibly losing on some rare signatures. The strategy used by C4.5rules starts with rules obtained from an overfitted decision tree, and then uses the entire training set for generalization of each rule. Generalization is guided by pessimistic error rate estimates. However, the estimate for a small disjunct may not be reliable because of its low support. So, any decision made by comparing this estimate to the estimate of its generalized version, may be unreliable.

As for the problem of splintered false positives, RIPPER may face it because of its learning method described earlier. C4.5 decision tree induction technique may also run into this problem. The greedy approach used by C4.5 splits the data-set after learning a decision at each node. If these decisions correspond to the signatures of the presence of target-classes, then the examples that are required to learn the absence of non-target-class will get split into multiple disjoint paths of the tree. C4.5rules starts with the initial rule-set supplied by the decision tree. Although its rule-pruning procedure usually improves the generalization ability of the C4.5 tree model, it has no mechanism to gather the splintered false positives and re-learn the signatures of absence of non-target-class present in them. Hence, we believe, C4.5rules may also face the same problem.

1.2 Paper Contributions

Here are the key contributions of this paper:

- Focus on complete and precise modeling of rare classes; i.e., the goal is shifted to achieving high recall as well as high precision for the given rare class as against achieving overall high accuracy.
- An improved version of the two-phase rule induction algorithm of [1] that allows a better implicit control over recall and precision.
- Illustrate the strengths of two-phase rule induction algorithm by designing various synthetic data models that range from relatively simple to fairly generic and complex to represent real-life classification scenarios.
- Empirical illustration, using synthetic models, of situations in which existing core rule-induction algorithms, C4.5rules and RIPPER, learn a very poor or unacceptable model. Our two-phase rule-induction algorithm performs very well in all these situations.
- Comparative results on a challenging real-life data-set that illustrate the effectiveness of the new improved version of the two-phase algorithm in yielding even better performance than its previous version as well as C4.5rules and RIPPER.

2. TWO PHASE RULE INDUCTION (PNRULE) ALGORITHM

We have recently proposed a classification framework based on the two-phase rule-induction approach [1]. Here, we give a brief overview of the key stages of the two-phase algorithm in the context of learning a binary classifier model for the specified target class.

We primarily describe some new parameters and methods that we have incorporated in the PNrule framework since the previously proposed version. In particular, the process of adding and growing a rule in P- and N-phases has new parameters that give user a control over recall and precision. Also, we have a new efficient algorithm to evaluate range-based conditions on numerical attributes.

We start with an overview of the main learning algorithm.

2.1 Main Learning Algorithm and Model Format

Given a training data-set and the target class, the algorithm learns a binary two-phase model for the target class. The model is represented using two kinds of rules: P-rules and N-rules. P-rules predict presence of the target class, whereas N-rules predict absence of the target class. P- and N-rules are learned using two sequential covering rule-learning stages. The true and false positives covered by high support and possibly low accuracy P-rules, are collected together before starting the N-phase. The two phases are followed by a step that constructs the scoring mechanism for P-N rule combinations. An overview of rule building and the scoring mechanism are given in following subsections. Some points to note regarding the overall algorithm are:

- The first phase of the algorithm (P-phase) strives for rules that have high support and reasonable accuracy. The mechanism used to build the rules, which includes the evaluation metric and the criterion for stopping the growth of a rule, and a user-specified minimum support together ensure that the rule's support does not become too low.
- Each stage (P-stage and N-stage) of the algorithm can have a distinct stopping criterion which decides when to stop adding more rules to the rule-set of that phase. In our current implementation, P-rules are added until a minimum user-specified fraction of the target class is

covered, and after that a new P-rule is added only if it satisfies a minimum accuracy threshold. If the desired minimum coverage is too high, some very low accuracy or low support rules may get added to the model. If the coverage is too low, some important rarer subclasses might be missed. So, a proper value of this parameter needs to found by experimentation. Currently, N-rules are added until the new rule increases the description length within some limit of the minimum value obtained so far [5].

- The final step of building the scoring mechanism is crucial. Without it, the model learned by PNrule framework will simply mean that if some P-rule applies and no N-rule applies to a record, then the record belongs to the target class C. Formally, this means $C = (P_0 \vee P_1 \vee ... \vee P_{n_P-1}) \wedge \neg N_0 \wedge \neg N_1 \wedge ... \wedge \neg N_{n_N-1}$, which is equivalently a DNF model of the form $C = (P_0 \wedge \neg N_0 \wedge \neg N_1 \wedge ... \wedge \neg N_{n_N-1}) \vee (P_1 \wedge \neg N_0 \wedge \neg N_1 \wedge ... \wedge \neg N_{n_N-1})$ $\vee ... \vee (P_{n_P-1} \wedge \neg N_0 \wedge \neg N_1 \wedge ... \wedge \neg N_{n_N-1})$. As can be seen, this model is restrictive in the sense that all conjunctions have all but one conditions in common. This might seem to restrict the kinds of functions that can be learned by our two-phase model. However, our scoring mechanism allows to relax this restriction, by selectively deciding to ignore the effects of certain N-rules on a given P-rule.

2.2 Building and Evaluating Rules

Each rule is built by discovering one conjunctive condition at a time. The candidate conditions used for categorical attributes are currently based on a single value of the attribute. For each numerical attribute, three kinds of conditions are evaluated. Two one-sided conditions, A $\leq v$ and A $> v$, are evaluated by a single scan of the data-set sorted on A's values. Unlike most existing techniques of rule-induction, we also explicitly evaluate a range-based condition $vl <$ A $\leq vr$. The values of vl and vr are computed by doing an extra scan of the sorted data-set. If condition A $\leq vr$ has higher value than condition A $> vl$, then we fix vr and scan for the best value of vl to the left of vr. If condition A $> vl$ has higher value than condition A $\leq vr$, then we fix vl and scan for the best value of vr to the right of vl. By experimentation, we have verified the ability of this simple yet efficient method to find close to best range-based conditions. A range-based decision can be emulated by two single-sided decisions; however, we believe that finding range-based conditions in one step has an advantage over methods that use only single-sided decisions. After discovering one of the limits of the range, the condition for other limit may not be discovered next or may not be discovered at all, if some attribute or condition becomes more prevalent.

There are two more features of our rule-building procedure that need explanation. First is evaluation metric. This metric is crucial in deciding which rules are added. The metric is expected to yield high value for a rule having high support and high accuracy with respect to the target class. The metric that we use in our experiments is **Z-number** [1], however other metrics can also be used. Possible choices are gini index, information gain, gain-ratio, chi-squared statistics or the metrics given in [13].

Final important aspect of rule building is the criterion for stopping the refinement or growth of the rule. Let current rule be R and the new refined rule be R1. We need to decide whether to accept R1 or to stop at R. R1 is a more specific version of R, so its accuracy is better than or equal to R, and its support is less than or equal to R. Based on the support and accuracy of R and R1, we find the values of the evaluation metric for both rules with respect to the distribution of target class in the data-set that remains after removing data supported by earlier rules. In P-phase, we accept R1 only if its evaluation metric value is more than that of R and its support is higher than the minimum specified fraction of the target class population. In N-phase also, we select R1 if its evaluation metric has higher value than R. However, N-phase is guided by a lower limit on the recall of the original target class (user-specified). If R1 has lower value than R, but choosing R will cause the recall to go below the lower limit, then we let the rule to be refined to R1. If this lower limit is too high, then lot of highly refined, low support rules might be discovered, leading to overfitting in N-phase and loss in overall precision. If its too low, then some N-rules may remain too general thus introducing lot of false negatives and thereby losing the recall. Note that the minimum coverage requirement in the P-phase (section 2.1) acts as an upper limit on the recall. Thus, these two parameters give user a control over classifier's recall and precision performance.

2.3 PNrule Classification Strategy and Scoring Algorithm

First we describe how we use P-rules and N-rules to classify an unseen record. P-rules and N-rules are arranged in decreasing order of significance, which is the same as their order of discovery. Given a record consisting of attribute-value pairs, P-rules are applied in their ranked order. If no P-rule applies, prediction is False. The first P-rule that applies is accepted, and then the N-rules are applied in their ranked order. The first N-rule that applies is accepted. We always have a default last N-rule that applies when none of the discovered N-rules apply. If the classifier has to make a simple True-False decision, then we can predict a record to be True only when some P-rule applies and no N-rule applies. However, it is desirable to assign a score to the decision made by the classifier that can be interpreted as the probability of the given record belonging to the target class. Hence, depending on which P-rule and N-rule combination applies, we predict the record to be True with certain score in the interval (0%,100%). For binary classification, we declare the record to be true if score is greater than some given threshold, usually 50%.

Another motivation for assigning a probabilistic score to each individual P-rule, N-rule combination is to weigh the effect of each N-rule on each P-rule. Remember that the N-rules were learned on a set of records *collectively* supported by all P-rules. So, each N-rule is significant in removing the collective false positives. However, a given N-rule may be effective in removing false positives of only a subset of P-rules. Moreover, some low accuracy N-rule may be introducing excessive false negatives for some P-rules, possibly because its primary contribution is to remove false positives of other lower accuracy P-rules. Such excessive false negatives can be recovered by assigning them a correspondingly low score. Thus, we need to properly judge the significance of each N-rule for each P-rule. Our scoring mechanism precisely does this. Detailed scoring algorithm is given in [1].

The overall effect of scoring mechanism is to selectively ignore effects of certain N-rules on a given P-rule. At the end of it all, ScoreMatrix reflects an approximate probability that a record belongs to the target class, given that a particular P-rule, N-rule combination applied to it.

3. ANALYSIS USING SYNTHETIC MODELS

The hypothesis that PNrule is suitable for rare classes, is empirically evaluated in this section by designing some synthetic datasets. We compare PNrule's performance with two other core, state-of-the-art rule-induction methods; viz. C4.5rules and RIPPER.

3.1 Comparison Strategy

We compare the *binary classification* performance of the methods. Our goal is to learn rule-based model to distinguish one class, the target class, from the rest.

In the comparative study that we present now, we use the following comparison metric. which is widely used by the information retrieval community [14]. Let target class C have p examples in a data-set. Let the classifier predict q out of these p examples correctly. The classifier also predicts r examples to be of class C whereas they actually belong to some other class (false positives). Then, *recall* is defined as $R = q/p$ and *precision* is defined as $P = q/(q + r)$. The comparison metric, F-measure, is defined as $F = 2RP/(R + P)$. This metric has a range of [0,1]. It is a special case of the general metric derived in [14], in which recall and precision are given equal weights. Higher values of F indicate that the classifier is performing better on recall as well as precision.

Some variations of each method are tried on each dataset, and the one which yields best results on the test data, is chosen. For PNrule, we used four combinations of values of upper limit on recall (0.95, 0.99) and lower limit on recall (0.7, 0.95). The rest of the input parameters of PNrule were fixed to very conservative values.

Most recent available versions of RIPPER (version 2.5) and C4.5, C4.5rules (Release 8) are used for comparison. For both, we use their default recommended settings of all input parameters. Two variations of these methods are obtained by using different training sets. One variation refers to the set of results when each record in the training set has unit weight. The second variation, denoted henceforth with extension of "-we", refers to the set of results with stratified training set[4] In such stratified set, each target class record has identical weight such that the sum of these weights is equal to the number of non-target-class records, each of which is given a unit weight. Stratification process converts an originally rare class into a class of equal strength.

3.2 Designing Datasets and Analysis of Performance

The purpose of designing synthetic datasets is to identify the nature of datasets where PNrule has an advantage over its competitors. The goal is to find situations that are general enough to resemble real-life scenarios, and PNrule's approach not only performs better but also becomes a necessity.

[4]For C4.5rules-we, we used stratified set to build an overfitted decision tree and this tree was used to construct rules using unit-weight training set.

The design process is guided by two strengths of PNrule:

- **Presence of second phase:**
  PNrule requires to go into second phase, only when it cannot find a set of very highly accurate and high support rules that cover almost entire target class. In other words, second phase is required whenever a high support P-rule *inevitably* captures a chunk of negative examples along with positive examples.

- **Collective removal of false positives in second phase:**
  PNrule collects all the false positives covered by the union of P-rules and learns N-rules on this collection to remove false positives. We believe that this mechanism makes PNrule less sensitive to the problem of splintered false positives, that its competitors may face as indicated in section 1. Note that the key difference made by the N-phase is that PNrule needs to learn the signatures the *presence* of non-target-class in this phase, unlike RIPPER and C4.5rules, which need to learn signatures for the *absence* of target class in attributes that distinguish non-target-class.

With these two strengths of PNrule in mind, we designed the synthetic datasets presented below. The first dataset is built from a fairly simple model that has only continuous-valued attributes. By varying the parameters of the model, we show how the performance of C4.5rules and RIPPER starts to deteriorate. Then, we extend these results to build a model that has only categorical-valued attributes. The final set of models is designed with a mix of continuous and categorical attributes and is made reasonably complex to resemble real-life situations. On this set of very general datasets, we also show the effect of varying the proportion of target class, and see why PNrule is indeed required for rare classes.

3.2.1 Datasets with only continuous-valued attributes

A simple model:
In the first simple model, both target class and non-target-class have multiple subclasses. Each subclass is distinguished by signatures on a single attribute. Signatures appear as disjoint, uniformly-spaced, identical peaks in the subclass distribution over the values of its distinguishing attribute. Records of other subclasses are randomly uniformly distributed over this attribute. The training examples of a given subclass are is equally divided among its disjoint signatures.

A pictorial description of one of the datasets, **nsyn3** given in Figure 1 shows how each class is distributed along every attribute. The first attribute has four distinguishing signature peaks for the target-class, whereas second and third attributes have similar signatures to distinguish NC1 and NC2, two subclasses of the non-target-class NC, respectively. The difficulty of the problem can be seen from how tiny and deeply embedded the signatures of C are in first attribute's distribution. Full coverage of C inherently captures large number of false positives. So, the classifier has to be able to learn good model for C as well as NC. The desired classifier model for the target-class in this dataset is as follows: C is true if the first attribute (from left) has a value in the range of one of the peaks of C AND the other two attributes have values that do not fall in the range of the signature peaks of

NC1 and NC2. In other words, in an ideal PNrule model for C, each of the C's peaks in first attribute form P-rules, and each of the peaks of NC1 and NC2 in other two attributes form N-rules.

The model has following parameters:

- Number of target subclasses (tc).
- Number of disjoint signatures per target subclass ($nsptc$).
- Total width of signature peaks for each subclass of target-class (tr). Increasing the value of tr increases the number of false positives that need to be removed in order to learn a high precision model.
- Number of non-target subclasses (ntc).
- Number of disjoint signatures per non-target subclass ($nspntc$).
- Total width of signature peaks for each subclass of non-target-class (nr).
- Shape of a signature's distribution ($d - shape$). This can be flat rectangular signifying a uniform distribution, or triangular, or Gaussian.

Table 1 describes various parameters used in the experiments conducted using this model, and gives comparative results for the three techniques on them. For each dataset, the training set has the target class population of 1,500 (0.3%) out of total 500,000 training set records. Thus, we are testing the techniques on a very rare class. The training set and test set are both generated from identical models. The results reported are on the test set, that has total of 250,000 records, among which 750 are that of target class.

The effect of varying $nsptc$, ntc, and $nspntc$ can be seen from Table 1, whereas the effect of variations in tr and nr can be seen from Figure 1 and Table 2.

The performance results in Table 1 indicate that as the number of signatures and number of subclasses of the non-target-class increase, the performance of C4.5rules and RIPPER starts to deteriorate. Observing the model learned by C4.5rules indicated that it learns rules capture the non-signature regions of values in the non-target-class attributes. For example, for dataset **nsyn2**, it learns 16 strong rules for target class. There are 2 subclasses of non-target-class and each subclass has three signatures, thus making up four non-signature regions. A disjunction of 16 possible combinations of these regions forms the rule-set for target class. This works as long as there are sufficient number of target-class examples in each such region. However, as the number of these combinations increases (for example to 125 in dataset **nsyn5**), C4.5rules performance dwindles. In fact, for this dataset, it is unable to learn an acceptable model (very poor recall as well as precision levels). The rules that C4.5rules learns for the non-target class are precisely the non-signature regions in the target-class attribute. Detailed look at the decision tree model from which C4.5rules gets its initial rule-set showed that the tree model is facing precisely the problem of splintering examples that was pointed out in section 3.2. Due to the lack of ability to collectively remove the false positives, it tries to remove false positives in each peak of the target-class individually. In the process, it learns incomplete description for the signatures of non-target class.

Now, let us see why RIPPER's performance degrades. Observation of its learned model indicated that it also faces the adverse effect of splintering. It learns correct signatures for the target class as the first conditions in the rules. But, since these signatures are impure (dominated by negative examples), it tries to remove these negative examples immediately by refining the rule. Like C4.5rules, RIPPER also has to add conditions that learn the non-signature regions of the non-target-class attributes. RIPPER's rules, however, are a more specific version of rules learned by C4.5rules, because most of RIPPER's rules have a target-class signature in them. So, as the combinations of non-signature regions of non-target class increases, RIPPER's performance starts to degrade earlier than that of C4.5rules.

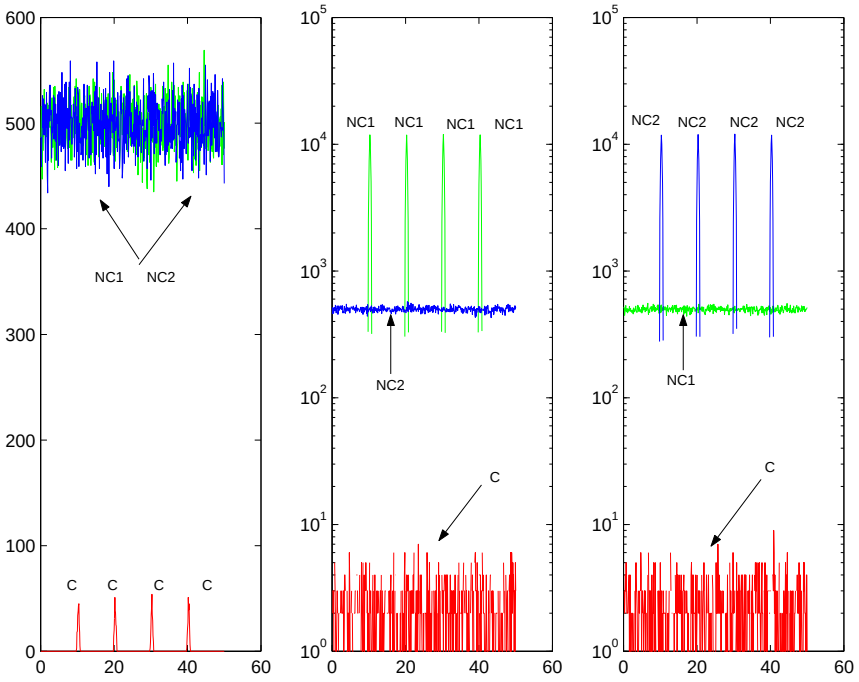
Looking at the PNrule results in Table 1, it can be seen that PNrule is able to learn good models in all the situations. In fact, an observation of the P-rules and N-rules learned by PNrule shows that it is learning close-to-precise signatures of target-class and is able to remove large number of false positives in the N-phase by learning close-to-precise signatures of the non-target-class. The strategy of stopping the refinement of P-rule when the global value does not improve, is working very well.

Effect of stratified training set:
With the stratified set, each target class record has high weight. The non-signature regions for non-target-class attributes now become much purer in target class as compared to unit-weight training set. This in conjunction with dominating signature peaks of the target-class attribute makes both C4.5-we and RIPPER-we to be satisfied with learning these regions in a piece-meal fashion. Thus, in effect they are learning an incomplete description to remove false positives. This results in a very high recall but poor precision, as can be observed in Figure 1 for dataset **nsyn3**. Note that we are reporting results of C4.5-we (tree model) and not C4.5rules-we (rule model). This is because usually C4.5-we models were very large trees, and it took unacceptable amount of learning time to build a rule model from them.

Effect of varying tr and nr:
Finally, let us analyze the effect of variation in tr and nr values. Results in Figure 1 for dataset **nsyn3** and Table 2 for dataset **nsyn5** are used for illustration.

An increase in the value of tr causes each signature peak to widen. When a correct signature of the target class is learned, this causes more false positives to be covered because non-target-class is uniformly distributed over target-class attributes. This is expected to have a good effect on techniques that suffer from the splintering problem, because now capturing a single target-class signature makes more negative examples available to the learner thereby increasing its chance of learning correct combinations of non-signature regions of the non-target-class. This effect can be observed in Figure 1 for C4.5rules and RIPPER, especially for values of $nr = 0.2$ and $nr = 2.0$.

As the value of nr increases, it becomes difficult to remove the false positives without removing more and more true positives of the target-class. This can be seen in the degradation of performance in all three methods using unit weight training sets. Of course, the degrading effect is more significant for C4.5rules and RIPPER than for PNrule. This can be attributed again to the splintering effect observed in C4.5rules, because an increase in nr causes a decrease in the number of examples available to learn the combinations



dataset: **nsyn3**												
	$nr = 0.2$				$nr = 2.0$				$nr = 4.0$			
	M	*Rec*	*Prec*	*F*	M	*Rec*	*Prec*	*F*	M	*Rec*	*Prec*	*F*
$tr =$ 0.2	C	97.07	98.78	**.9792**	C	84.31	89.42	.8679	C	36.84	60.75	.4586
	Cte	97.61	28.86	.4455	Cte	98.67	30.34	.4640	Cte	98.27	29.34	.4518
	R	68.88	73.16	.7096	R	42.55	65.71	.5165	R	27.93	55.41	.3714
	Re	95.74	30.78	.4659	Re	98.94	30.67	.4682	Re	98.01	30.56	.4659
	P	95.21	99.44	**.9728**	P	84.44	97.54	**.9052**	P	68.75	95.04	**.7978**
$tr =$ 2.0	C	92.69	99.71	**.9607**	C	77.79	98.32	**.8686**	C	48.40	97.07	.6460
	Cte	59.44	5.54	.1013	Cte	57.31	4.78	.0882	Cte	61.97	4.90	.0908
	R	78.99	99.83	.8820	R	34.57	96.30	.5088	R	2.53	73.08	.0488
	Re	94.95	5.88	.1108	Re	94.68	4.44	.0849	Re	98.54	4.12	.0791
	P	89.89	98.11	.9382	P	78.46	96.88	**.8670**	P	67.15	94.75	**.7860**
$tr =$ 4.0	C	92.15	99.86	.9585	C	75.66	99.13	.8582	C	39.49	96.43	.5604
	Cte	57.85	4.30	.0801	Cte	52.13	3.83	.0714	Cte	49.34	3.27	.0613
	R	73.01	100.0	.8440	R	46.01	93.77	.6173	R	7.18	94.74	.1335
	Re	88.16	7.37	.1360	Re	96.41	2.21	.0432	Re	92.95	2.29	.0447
	P	95.08	99.44	**.9721**	P	78.86	99.16	**.8785**	P	66.89	91.12	**.7715**

Figure 1: Effect of varying tr and nr on dataset nsyn3. Top part gives the pictorial description of the distributions of target class C and two subclass (NC1 and NC2) of the non-target class, over the values of three numerical attributes. The X-axes have values of the attributes, Y-axes show the number of examples of each class. First attribute (leftmost graph) has distinguishing signatures (peaks in the distribution) for C, because NC1 and NC2 are uniformly and randomly distributed. Similarly, second and third attributes have signatures for NC1 and NC2. For each combination of tr and nr, the best classifiers are indicated by bold-faced F values. Classifier notations are C: C4.5rules, Cte: C4.5-we (tree model), R: RIPPER, Re: RIPPER-we, P: PNrule

numerical-only datasets							
	target class			non-target class			
dataset	tc	$nsptc$	tr	ntc	$nspntc$	nr	$d-shape$
nsyn1	1	1	0.2	2	3	0.2	triangular
nsyn2	1	4	0.2	2	3	0.2	triangular
nsyn3	1	4	0.2	2	4	0.2	triangular
nsyn4	1	4	0.2	2	5	0.2	triangular
nsyn5	1	4	0.2	3	4	0.2	triangular
nsyn6	1	4	0.2	3	5	0.2	triangular

dataset	C4.5			C4.5-we			RIPPER			RIPPER-we			PNrule		
	Rec	$Prec$	F	Rec	$Prec$	F	Rec	$Prec$	F	Rec	$Prec$	F	Rec	$Prec$	F
nsyn1	97.20	99.73	**.9845**	98.93	29.11	.4498	96.13	99.86	**.9796**	99.60	35.02	.5182	98.13	99.73	**.9892**
nsyn2	94.81	99.72	**.9721**	98.67	30.27	.4633	89.63	99.70	.9440	94.41	39.60	.5580	94.81	99.30	**.9701**
nsyn3	97.07	98.78	**.9792**	97.61	28.86	.4455	68.88	73.16	.7096	95.74	30.78	.4659	95.21	99.44	**.9728**
nsyn4	90.82	99.13	.9480	96.81	29.35	.4505	35.51	58.04	.4406	94.68	34.45	.5051	96.41	97.45	**.9693**
nsyn5	7.05	54.64	.1249	96.94	29.13	.4479	31.91	44.86	.3730	96.01	29.66	.4532	94.28	97.93	**.9607**
nsyn6	6.78	49.51	.1193	96.94	29.04	.4470	7.45	50.91	.1299	96.54	29.84	.4559	92.69	97.21	**.9489**

Table 1: Comparative Results on numerical-only datasets. First table gives description of each data-set. See text for an explanation of column names. Rec: recall (R), Prec: precision (P), F: 2RP/(R+P). Bold-faced values of F indicate the classifiers which perform best for the given dataset.

of non-signature regions of non-target-class. PNrule is still able to learn models with reasonably acceptable levels of recall and precision.

Effect of increasing tr on C4.5-we and RIPPER-we is to incur further loss in precision. This is due to even wider highly weighted target-class peaks. The increased width causes more false positives to be covered by each target class signature, and increased weight of true positives in each signature misleads these techniques to further ignore the job of removing the false positives.

Performance of C4.5-we and RIPPER-we is more or less unaffected by the value of nr, in fact it may increase slightly with an increase in nr. This can be explained as follows. As nr increases, the non-signature regions in non-target-class shrink. Because the target class examples have much higher weights in RIPPER-we, these regions become purer. Thus they can be learned a bit more accurately.

Summary:
In summary, for all the datasets generated with this simple disjunctive model, when C4.5rules and RIPPER can learn good models, PNrule can also learn equally good models. However, C4.5rules and RIPPER start to fall apart as the learning difficulty increases; and PNrule still holds its performance to an acceptably high level.

3.2.2 Datasets with only categorical-valued attributes

Extending the idea of designing numerical-only datasets, we can create a model to generate datasets that have only categorical attributes. Instead of representing signatures as peaks in the distribution over a continuous-valued attribute, we form each signature by a conjuction of a set of words. Figure 2 gives the description of categorical-only datasets that are generated. The role of tc and ntc is played by distinct values of na for target and non-target class. The role of $nsptc$ and $nspntc$ is played by $nspa$, and roles of tr and nr are played by values of $nwps$ for target and non-target-class.

The results on these categorical-only datasets, tabulated in Table 3, are similar to those seen on the numerical-only datasets in the sense that PNrule outperforms both RIPPER and C4.5rules. The results for datasets of categories A and B indicate that, as the number of signatures of non-target-class increases ($na \times nspa$), performance of PNrule's competitors degrades. This observation is similar to that made for numerical-only datasets. The effect of variation in the probability of each signature can be seen in datasets of category C. Once again, as the probability increases, performance of all the classifiers decreases. However, the decrease in PNrule's performance is still at an acceptable level.

3.2.3 General Datasets

Finally, one very generic dataset is generated. This dataset has both categorical and numerical attributes, each attribute distinguishing one subclass of target or non-target class. The description of this dataset, **syngen**, is given in Figure 3. The signature of the subclass C1 of the target-class is formed by a disjunction of two conjunctions of its peaks (marked C1) in first two attributes (left two graphs in the Figure). Note that these two attributes also have signatures for a subclass NC1 of non-target-class. Signatures for subclasses C2 and NC2 of target and non-target class, are disjunctions of their respective peaks (last two graphs). Subclasses C3 and NC3 are distinguished by categorical attributes having signature parameters shown in the figure. We believe that this model is fairly general and complex to represent real-life situations. The results on this generic model are given in Table 4. They show that PNrule outperforms its best competitor techniques, on all the combinations of tr and nr, thus illustrating PNrule's strength on a complex dataset.

3.3 Why PNrule is especially better for Rare Classes

After demonstrating PNrule's superiority in many different situations, including complex close-to-reality scenarios, it would be interesting to see why we claim PNrule to be especially suitable for rare classes. In all the datasets that were described so far, the target class size was set to 0.3% in the training dataset. Now, we take the most general dataset that we designed, **syngen**. Keeping its data generation model the same, we vary the percentage of target class examples available to the learner. This is done by taking the datasets (both training and test) with 0.3% target

dataset: **nsyn5**								
	nr=0.2				nr=4.0			
	M	*Rec*	*Prec*	*F*	M	*Rec*	*Prec*	*F*
tr= 0.2	Cte	96.94	29.13	.4479	Cte	97.47	30.57	.4654
	Re	96.01	29.66	.4532	Re	97.74	30.70	.4673
	P	94.28	97.93	**.9607**	P	67.02	80.00	**.7294**
tr= 4.0	Cte	39.76	2.66	.0499	Cte	37.23	2.50	.0469
	Re	86.97	2.61	.0507	Re	98.27	2.11	.0413
	P	93.35	96.56	**.9493**	P	57.18	57.03	**.5710**

Table 2: Comparative Results on dataset nsyn5. Cte: C4.5-we, Re: RIPPER-we

na: Each class has *na* number of subclasses.
nspa: Each subclass is distinguished by *nspa* number of disjoint signatures over a distinct pair of attributes.
nwps: Each distinct pair of attributes has total *nwps* combinations of words that identify the corresponding signature.
Each subclass of the target-class has same values of *nspa* and *nwps*, but different set of distinguishing words.
Each subclass of the non-target-class has same values of *nspa* and *nwps*, but different set of distinguishing words.

Example::

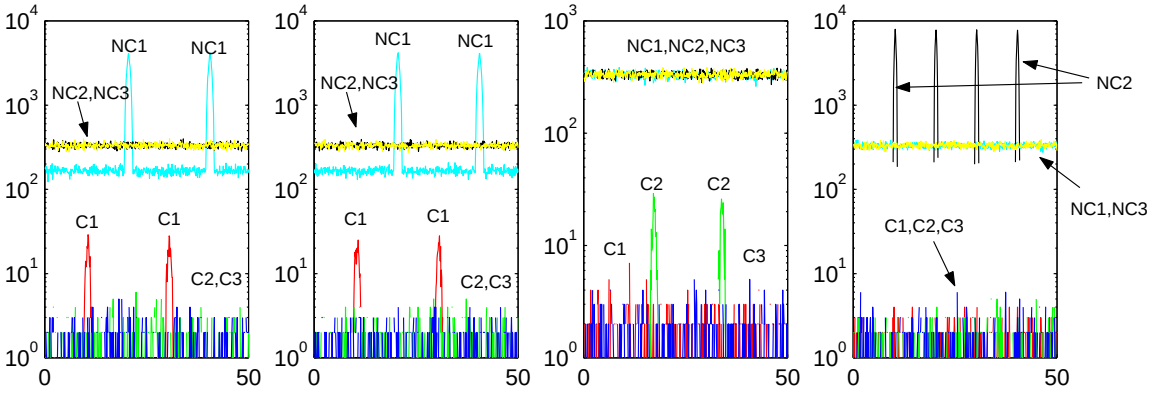
target-class: $na = 1$, $nspa = 2$, $nwps = 4$ (2x2);
subclass C1: Csig11 OR Csig12;
Csig11:(AC11=w1111 AND AC12=w1211) OR (AC11=w1111 AND AC12=w1212) OR
(AC11=w1112 AND AC12=w1211) OR (AC11=w1112 AND AC12=w1212);
Csig12:(AC11=w1121 AND AC12=w1221) OR (AC11=w1121 AND AC12=w1222) OR
(AC11=w1122 AND AC12=w1221) OR (AC11=w1122 AND AC12=w1222);

Figure 2: Description of categorical-only datasets

categorical-only datasets							
	dataset	target class			non-target class		
		na	nspa	nwps	na	nspa	nwps
A	coa1	1	3	2/400	2	3	2/100
	coa2	1	3	2/400	3	3	2/100
	coa3	1	3	2/400	4	3	2/100
B	coa4	1	4	2/400	2	4	2/100
	coa5	1	4	2/400	3	4	2/100
	coa6	1	4	2/400	4	4	2/100
C	coad1	2	4	2/400	4	4	2/400
	coad2	2	4	2/400	4	4	2/100
	coad3	2	4	2/100	4	4	2/400
	coad4	2	4	2/100	4	4	2/100

	dataset	C4.5rules			RIPPER			PNrule		
		Rec	*Prec*	*F*	*Rec*	*Prec*	*F*	*Rec*	*Prec*	*F*
A	coa1	82.40	100.0	**.9035**	100.0	16.74	.2868	73.33	100.0	.8462
	coa2	62.93	100.0	.7725	100.0	16.90	.2892	83.20	100.0	**.9083**
	coa3	46.13	99.14	.6297	100.0	16.79	.2875	78.40	100.0	**.8789**
B	coa4	72.21	100.0	.8386	100.0	13.13	.2321	85.11	100.0	**.9195**
	coa5	42.69	100.0	.5983	100.0	13.09	.2316	76.86	100.0	**.8692**
	coa6	22.74	97.16	.3685	100.0	13.16	.2326	71.28	100.0	**.8323**
C	coad1	11.44	13.98	.1258	100.0	7.04	.1315	89.23	65.40	**.7548**
	coad2	100.0	0.30	.0060	100.0	7.10	.1325	65.69	51.24	**.5758**
	coad3	5.05	35.51	.0885	100.0	1.93	.0379	64.23	84.15	**.7285**
	coad4	20.88	100.0	.3454	100.0	1.92	.0377	72.07	100.0	**.8377**

Table 3: Comparative Results on categorical-only datasets. First table gives description of each data-set. See text for an explanation of column names.



C3 and NC3, subclasses of target and non−target respectively,
are distinguished by categorical attributes:

C3: na=1, nspa=2, nwps=2

NC3: na=1, nspa=4, nwps=2

Figure 3: Description of dataset syngen. There are 8 total attributes: four categorical attributes distinguish subclasses C3 and NC3 or target and non-target class, respectively. Four numerical attributes, over which the class distributions are shown, distinguish other subclasses. The distribution of a subclass over non-distinguishing attributes is random and uniform.

dataset: **syngen**								
	nr=0.2				nr=4.0			
	M	*Rec*	*Prec*	*F*	M	*Rec*	*Prec*	*F*
tr= 0.2	C	25.73	93.69	.4038	C	26.80	85.90	.4085
	Re	91.60	13.95	.2717	Re	96.67	14.92	.2586
	P	85.87	94.29	**.8988**	P	66.13	65.78	**.6596**
tr= 4.0	C	25.33	100.0	.4043	C	9.60	83.72	.1722
	Re	88.13	2.28	.0444	Re	95.07	2.30	.0450
	P	75.07	98.77	**.8530**	P	52.40	48.04	**.5013**

Table 4: Comparative Results on dataset syngen

dataset: **syngen (tr=0.2,nr=0.2)**				
ntc-frac	tc %	C4.5rules	RIPPER	PNrule
1.0	0.3%	.4038	.2717	**.8988**
0.5	0.6%	.5177	.4137	**.9208**
0.1	2.9%	.7569	.7766	**.9090**
0.05	5.7%	.8261	**.8643**	**.8709**
0.02	13.1%	**.9270**	**.9395**	**.9390**
0.01	23.1%	**.9448**	**.9644**	**.9603**
0.003	50.0%	.9577	**.9840**	.9539

dataset: **syngen (tr=4.0,nr=4.0)**				
ntc-frac	tc %	C4.5rules	RIPPER	PNrule
1.0	0.3%	0.1722	0.0450	**0.5013**
0.1	2.9%	0.5326	0.5293	**0.6181**
0.05	5.7%	0.6411	0.6639	**0.6944**
0.02	13.1%	0.6545	0.7314	**0.7598**
0.01	23.1%	0.7681	0.7935	**0.8328**

Table 5: Comparing effect of variation in target class size on syngen dataset. ntc-frac is the fraction of non-target-class examples randomly sampled from the dataset with 0.3% target class proportion.

Class	C4.5rules			RIPPER			old PNrule [1]		
	Rec	Prec	F	Rec	Prec	F	Rec	Prec	F
probe	73.04	86.38	0.7915	81.16	77.92	0.7951	89.01	82.11	**0.8542**
r2l	5.23	96.36	0.0993	8.33	81.85	0.1512	13.05	82.37	**0.2252**

Table 6: Results of C4.5rules, RIPPER, and older version of PNrule for binary classification for r2l and probe in the KDDCUP'99 test data-set.

class proportion, retaining all the target class examples in them, and randomly sampling a varying fraction of the non-target-class examples. The results are given in Table 5.

As the target class proportion increases, the difference between the performances of all the three techniques becomes lesser and lesser. For the simpler **syngen** dataset (tr=0.2, nr=0.2), PNrule certainly has a definitive edge over its competitors for target class proportions of upto 5.0%, Moreover, PNrule also performs competitively when target class is not rare. For the tougher datasets, such as **syngen** with tr=4.0 and nr=4.0 as shown in Table 5, PNrule's edge can extend to even larger target class proportions.

Hence, PNrule is clearly the best choice when target class is rare. This final set of results on synthetic datasets corroborates our claim that for such classes, it is necessary to have PNrule's capability of attacking recall and precision goals separately by first learning high support rules to get recall, and later collectively removing the false positives that get inevitably covered.

4. PNRULE ON A REAL-LIFE DATASET

As illustrated by synthetic datasets, PNrule's two-phase design certainly has a potential to perform well on rare classes. In this section, we test the technique on an actual real-life dataset.

We use a dataset from the domain of network intrusion detection. It was provided as part of the KDD-CUP-99 classifier learning contest [7]. The data-set is collected by monitoring a real-life military computer network that was intentionally peppered with various attacks that hackers would use to break in. Original training set has close to 5 million records belonging to four attack classes and one no-attack or normal class. A 10% sample of this original set was also supplied as part of the contest. We use this sample, $T_{10\%}$, in our experiment. We had reported some results on the rare classes of this data-set using the previous version of PNrule [1]. Our interest in this paper is to study the effect of the new parameters of PNrule, especially the minimum coverage of target class in P-phase and lower recall limit used in N-phase, on improving the PNrule performance further. We present results for two rare classes: probe and r2l, whose respective populations in the $T_{10\%}$ training data-set are 0.83% and 0.23%. The $T_{10\%}$ data-set preserves *all* the examples of these rare classes from the original training set.

The test data used here is the test-data supplied as part of the contest. Note that this test-data has a very different distribution of these classes (1.34% for probe and 5.2% for r2l) and some new subclasses that are not present in the training data, so there is some inherent limitation on how good the core data mining techniques such as PNrule, C4.5rules, and RIPPER can perform.

As with synthetic datasets, we used default settings for RIPPER and C4.5, but we tested them with two training sets: as-is and stratified. The best results of these classifiers on the test dataset are given in Table 6.

The results we had obtained with our previous version of PNrule are shown in Table 6. These results are certainly better than C4.5rules and RIPPER. But, we want to see if we can obtain further improvement with the improved PNrule version of this paper. So, we start by varying two of its control parameters: minimum target class coverage in P-Phase (denoted *rp*) and lower limit on recall used to control growth of a rule in N-phase (denoted *rn*). We use RIPPER's evaluation metric of information gain for these experiments.

The results for r2l are as follows, where R, P, and F denote recall, precision, and F-measure respectively:

r2l		*rn*		
rp		0.8	0.95	0.995
0.95	R	6.10	6.02	6.02
	P	99.30	99.39	99.39
	F	.1149	.1135	.1135
0.995	R	6.63	6.63	8.58
	P	59.35	59.35	71.08
	F	.1192	.1192	**.1531**

The effect of increasing *rn* is negligible for value of *rp*=0.95. Whereas for *rp*=0.995, which covers almost the entire target class in first phase, thereby covering relatively more false positives, variation in *rn* has some role to play by controlling how refined the N-rules become. For large values of *rn*, N-phase is forced to refine N-rules to the point they cover very small number of false positives of N-phase (i.e. the original target class). As can be seen, the precision as well as recall get affected because of this. The best result here is still just similar to RIPPER. We observed that the P-rules together are covering relatively small number of false positives. This usually is a good thing, but given the skewed and different distribution of test-data, we decided to try making P-rules very general. Here is the effect of variation in *rp* and *rn* on the model learned, when P-rule length was kept at 1:

r2l.P1		*rn*		
rp		0.9	0.95	0.995
0.95	R	6.04	13.34	13.05
	P	98.59	83.11	82.37
	F	.1138	**.2299**	.2252
0.995	R	8.23	10.22	10.43
	P	98.23	99.04	98.54
	F	.1519	.1853	.1887

Following observation can be made from these results: Restricting P-rule length to 1 allows P-rules to be very general, thus giving PNrule more ability to *collectively* remove the false positives in second phase. Furthermore, by varying *rp* and *rn*, one can control the recall and precision obtainable from the classifier for the given class. We have improved substantially over RIPPER and C4.5rules for learning r2l's signatures.

Now, let us see how PNrule performs on probe by doing similar experiments. First, here are the results when P-rule length was controlled by the growth criterion given in section 2.2; i.e. it was not artificially restricted:

probe		*rn*		
rp		0.8	0.95	0.995
0.95	R	87.16	87.16	87.16
	P	74.64	74.64	74.64
	F	**.8041**	.8041	.8041
0.995	R	87.54	86.20	86.13
	P	73.32	68.54	72.81
	F	.7980	.7636	.7891

We can see similar effect of varying *rn* as was seen for r2l; i.e. it is not affecting the performance much. Again, we obtain results close to RIPPER's results. We observed that, as with r2l, the P-phase is covering relatively small number of false positives. So, we decided to make each P-rule very general. Here are the results when P-rule length is 1:

probe.P1		rn	
rp		0.9	0.995
0.95	R	84.69	87.37
	P	92.38	73.43
	F	**.8837**	.7980
0.995	R	84.69	87.37
	P	92.38	73.43
	F	.8837	.7980

The performance gets a sudden boost by using very general P-rules. Now, PNrule's margin of performance for probe has become much stronger over C4.5rules and RIPPER, as compared to our previous PNrule version. PNrule's performance on probe can be attributed to two things: first its ability to *collectively* remove false positives in second phase; and second, the ability to control recall and precision by varying rn and rp.

Result of probe.P1 and r2l.P1 show that sometimes it helps to generate more false positives in first phase. Also, all the above results indicate that choosing correct values of parameters rp and rn is important. If rn is set too high, as with rn=0.995 in probe.P1, then the classifier might gain somewhat on the recall but may lose precision significantly by overfitting the N-rules (i.e. not being able to remove false positives). Of course, if rn is too low, as in rn=0.9 in r2l.P1, then total recall will suffer and F-measure will be lower. If rp is set too high, as with rp=0.995 ($rn > 0.9$) for r2l.P1, then the later rules in P-phase may run into overfitting (most probably due to problem of small disjuncts), thus reducing the recall. If rp is set too low, as with rp=0.95 in the table of r2l with P-rule length not restricted artificially, then again the recall may suffer because of low target class coverage. In general, the combination of rp and rn determines the overall performance, along with their individual values.

All these observations and results illustrate that the two-phase approach and the control parameters of the improved version of PNrule indeed give extra tools and ability to outperform existing state-of-the-art techniques of its class, for efficient modeling of rare classes in challenging real-life situations such as this.

5. CONCLUSION

We empirically demonstrate the necessity of a two-phase rule-induction framework for learning effective rule-based models, especially when the target class being modeled has very small yet sufficient number of examples in the training data. Experiments on specially designed synthetic datasets and one real-life data-set show the strength and ability of our technique to perform very well for rare target classes. At the same time, via these data-sets, we identify and analyze the situations in which existing state-of-the-art classifiers fail to build an acceptable model. The primary contribution of this paper is to show that the method of attacking recall and precision separately in two phases has a potential to solve the problem of splintered false positives and problem of small disjuncts that the existing techniques may face especially while modeling rare classes.

The proposed framework opens up many avenues for further research. Some possibilities are: evaluating various stopping criteria for growing rules, automating or guiding the selection of recall limits in each stage, adding some pruning mechanisms to further protect the N-stage from running into overfitting, improving scoring mechanism to reflect close-to-true probabilities, and finally, extending the two-phase approach to a multi-phase approach.

6. REFERENCES

[1] R. C. Agarwal and M. V. Joshi. PNrule: 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*, Chicago, April 2001. Expanded version available as IBM Research Division Report, RC 21719, April 2000.

[2] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[3] P. Clark and T. Niblett. The CN2 induction algorithm. *Machine Learning*, 3:261–283, 1989.

[4] W. Cohen and Y. Singer. A simple, fast, and effective rule learner. In *Proc. of Annual Conference of American Association for Artificial Intelligence*, pages 335–342, 1999.

[5] W. W. Cohen. Fast effective rule induction. In *Proc. of Twelfth International Conference on Machine Learning*, Lake Tahoe, California, 1995.

[6] A. Danyluk and F. Provost. Small disjuncts in action: Learning to diagnose errors in the local loop of the telephone network. In *Proc. of Tenth International Conference on Machine Learning*, pages 81–88. Morgan Kaufmann, 1993.

[7] C. Elkan. Results of the KDD'99 classifier learning contest. In *http://www-cse.ucsd.edu/~elkan/clresults.html*, September 1999.

[8] R. C. Holte, L. Acker, and B. Porter. Concept learning and the problem of small disjuncts. In *Proc. of Eleventh International Joint Conference on Artificial Intelligence (IJCAI-89)*, pages 813–818, 1989.

[9] R. Michalski, I. Mozetic, J. Hong, and N. Lavrac. The multi-purpose incremental learning system AQ15 and its testing application to three medical domains. In *Proc. of Fifth National Conference on AI (AAAI-86)*, pages 1041–1045, Philadelphia, 1986.

[10] T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[11] J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[12] R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[13] Y.-S. Shih. Family of splitting criteria for classfication trees. *Statistics and Computing*, 9:309–315, 1999.

[14] C. J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 1979.

[15] G. M. Weiss. Learning with rare cases and small disjuncts. In *Proc. of Twelfth International Conference on Machine Learning*, pages 558–565, Lake Tahoe, California, 1995.

[16] S. M. Weiss and N. Indurkhya. Lightweight rule induction. In *Proc. of Seventh International Conference on Machine Learning (ICML-2000)*, 2000.