RC 21719 (97811) 7 April 2000 **ComputerScience/Mathematics**

IBM Research Report

PNrule: A New Framework for Learning Classifier Models in Data Mining (A Case-Study in Network Intrusion Detection)

Ramesh Agarwal, Mahesh V. Joshi
IBM T. J. Watson Research Center
P. O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - T. J. Watson - Tokyo - Zurich

PNrule: A New Framework for Learning Classifier Models in Data Mining (A Case-Study in Network Intrusion Detection)

Ramesh Agarwal[*] Mahesh V. Joshi[†]

March 2, 2000

Abstract

We have developed a new solution framework for the multi-class classification problem in data mining. The method is especially applicable in situations where different classes have widely different distributions in training data. We applied the technique to the Network Intrusion Detection Problem (KDD-CUP'99).

Our framework is based on a new rule-based classifier model for each target class. The proposed model consists of positive rules (P-rules) that predict presence of the class, and negative rules (N-rules) that predict absence of the class. The model is learned in two phases. The first phase discovers a few P-rules that capture most of the positive cases for the target class while keeping the false positive rate at a reasonable level. The goal of the second phase is to discover a few N-rules that remove most of the false positives introduced by the union of all P-rules while keeping the detection rate above an acceptable level. The sets of P- and N-rules are ranked according to certain statistical measures. We gather some statistics for P- and N-rules using the training data, and develop a mechanism to assign a score to each decision made by the classifier. This process is repeated for each target class. We use the misclassification cost matrix to consolidate the scores from all binary classifiers in arriving at the final decision. In this paper, we describe the details of this proposed framework.

A real-life network intrusion-detection dataset was supplied as part of the KDD-CUP'99 contest. This dataset of 5 million training records has a very highly skewed distribution of classes (largest class has 80% of the records, while the smallest has only 0.001% records). We describe how we applied our framework to this problem. As an aside, we also describe the controversy that we triggered after the contest and how we proved the original test data labels to be wrong. We compare the results of our approach with 23 other contestants. For the subset of test data consisting of known subclass labels, our technique achieves the best performance of all in terms of accuracy as well as misclassification cost penalty.

[*] IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598 **(agarwal@watson.ibm.com)**

[†] Department of Computer Science, University of Minnesota, Minneapolis, MN 55455 **(mjoshi@cs.umn.edu)**. This work was done when the author visited IBM Research during Summer'99.

1 Introduction and Motivation

Learning classifier models is an important problem in data mining. Observations are often 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, the problem is to learn a model for 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. Traditionally, the goal has been to minimize the number of misclassified records; i.e. to maximize accuracy.

Various techniques exist today to build accurate classifier models. These have emerged from research in many fields such as machine learning, neural networks, pattern recognition, statistics, etc [10, 19, 3]. Although no single technique is proven to be the best in all situations, techniques that learn rule-based models are especially popular in the domain of data mining. Even other forms of models such as decision trees or neural networks, are often post-processed to build a rule-based model. This can be contributed to the easy interpretability of the rules by humans, and competitive performance exhibited by rule-based models in many application domains.

A general rule-based model is a set of conditions on attributes, arranged in a disjunctive normal form (DNF), which means it is a disjunction (or union) of rules where each rule is a conjunction (or intersection) of conditions imposed on different attributes. Learning rule-based models directly from the training data has been studied in great detail in past couple of decades. The research so far has concentrated on developing techniques that intermix different search directions (general-to-specific or specific-to-general) with different performance metrics for rule evaluation and different stopping criteria for the search (we review them in Section 1.1 later). Their common goal is to discover small number of rules (low cardinality), which cover most of the positive examples of the target class (high coverage) and very few of the negative examples (high accuracy).

General-to-specific search techniques start with the most general rule, an empty rule, and progressively add specific conditions to it. When conjunction of conditions is added, the accuracy of the rule increases while its coverage and support decrease. Note that support is the total number records where rule applies, whereas coverage is the number of positive examples covered by the rule. An ideal situation is when exactly one conjunctive rule gives desired accuracy with entire coverage of the target class. But, this rarely happens in the real-world because usually a class consists of multiple subsets each with unique signatures. This especially occurs when a class consists of multiple distinct subclasses. Thus, disjunction of rules becomes a necessity. Disjunctions are discovered iteratively. In each iteration, a high accuracy conjunctive rule is discovered. Then the records covered by this rule are removed, and next iteration starts with the remaining examples. The tolerance on rule accuracy is usually very tight in each iteration. These are called sequential covering algorithms, and have found widespread use in rule-based modeling. However, they face a problem. As the algorithm proceeds, the data from which the rules are learned decreases in size. Hence, the support for the rules learned decreases. If the support is allowed to reduce substantially then the rules that are found may be too specific, thus overfitting the training data, or they may be overly general, because of the noise present in the data. Instead, if one stops the iterations after the remainder data size falls below some threshold, the rare subclasses might be missed. Rules with small coverage, learned from small datasets, are called *small disjuncts*. This was first identified by [14], in which they show that such rules tend to contribute more to the generalization error rate as compared to the large disjuncts (rules with high coverage). Detailed scenarios under which it can occur are discussed in [27, 6].

One remedy to avoid this problem is to use specific-to-general search techniques [7], which start with each record as the most specific rule, and progressively generalize the rule-set. Although it was shown [7] that such techniques have better ability to learn complex models involving many small disjuncts, they suffer because of too many specific conditions that they have to generalize. If each condition is to be generalized exhaustively, then their computational complexity scales poorly for large training data-sets (quadratic in training set size, and cubic in number of attributes [7]). An approach based on selecting few random positive examples to generalize, is given in [20], but it was shown to avoid exponential complexity only by imposing some restrictions on the types of rules learned while relying heavily on the background knowledge. Hence, specific-to-general type of techniques are not suitable for real-life problems in today's data mining applications involving large datasets.

So, let us concentrate on general-to-specific strategies, for which a few remedies are proposed in [14, 1] to solve the problem of small disjuncts so far. One remedy is to relax the emphasis on generality of rules, thus making them more specific for each of the iterations [14]. This has shown to reduce error rate of small disjuncts at the cost of increased error rate of large disjuncts. Another solution proposed by [1] is to assign probabilistic measures to the rules discovered in the hope of assigning lower measures to small disjuncts. Some other solutions proposed [22] are based on estimating a generalization accuracy from the accuracy in training data and use it to decide whether to retain or remove small disjuncts. But, all these solutions can be considered as workarounds for the actual problem, which is that of the trade-off between support and accuracy of the rules discovered. We suspect that the problem occurs because of relatively tight accuracy constraints used in all the iterations. This causes rules with small support to be discovered as algorithm progresses, thus leading to the problem. We believe that if accuracy constraints are relaxed gradually as required, then we can keep finding rules with sufficiently large support until most of the positive examples are covered. This is precisely the crux of our proposed approach.

In this paper, we propose a *two stage* general-to-specific framework of learning a rule-based model, which handles the small disjunct problem in a more prescriptive and effective manner as compared to existing methods. Also, our proposed method has a runtime that is linear in the number of training data records and linear in number of attributes, for each of its iterations. We call our framework PNrule, because it is based on finding rules that predict presence of a target class (P-rules) as well as rules that predict absence of a target class (N-rules). We handle the multi-class problems by learning binary classifiers for individual classes, which has been shown to be better [1] in handling small disjunct problems, than building a single model for all classes. Also, we think that this approach is especially desirable for small classes, which might be treated as noise by many performance metrics.

The key idea is to learn a set of P-rules that together cover positive examples with sufficient support such that each rule covers large number of examples to maintain its statistical significance. Initially, highly accurate rules are selected, but later accuracy is compromised in favor of support. What differentiates our method from all the previous approaches is its second stage. It learns N-rules that essentially remove the negative examples collectively covered by the union of all the P-rules. The existence of this second stage helps in making the algorithm less sensitive to the problem of small disjuncts in the first stage. Moreover, we rank each of the P- and N-rules, and use their accuracy and support statistics to assign a score to each decision made by the classifier.

One more feature of our proposed framework is that it is cost-sensitive. In other words, it is suitable for taking into account different costs of misclassifying different classes. For example, the cost of not identifying a mailing responder is certainly more than that of not identifying a non-responder in direct marketing applications. Various approaches have been proposed recently [25], either based on adjusting the distribution of training data according to the cost matrix (called stratification-based approaches) [13], or based on adding a cost-based framework on top of existing accuracy-driven classifiers [8]. Our framework uses scores generated by individual binary classifiers and the misclassification cost matrix, to arrive at predictions according to Bayes optimality rule for minimum expected cost.

In order to validate our proposed framework, we applied it to a real-life dataset from the network intrusion detection application. This data-set was supplied as a part of KDD-CUP'99 classifier contest [12]. The dataset contained about 5 million records, each with 41 attributes (34 continuous and 7 categorical), belonging to four attack classes and one normal (or no-attack) class. The contest was challenging in four respects. One, training dataset had a wide distribution of class populations (80% for the largest class and 0.001% for the smallest class). Second, misclassification cost was the evaluation criterion, with one of the smallest classes having the highest penalty for getting misclassified. Third, it was told beforehand that the test data-set on which the contestants will be evaluated has a different distribution of classes than training data-set. Finally, the four attack classes consisted of 39 different subclasses, out of which only 22 were present in the training data, the rest were present only in the test-data. We participated in this contest. As a matter of fact, the technique we are proposing in this paper evolved during and after our participation in the contest. We applied our proposed PNrule framework to this dataset. We compare our results with those of 23 other participants, hoping that many of them have used their own or other known state-of-the-art techniques. For the subset of test-data that belonged to subclasses that are present in the training data-set, our technique performs the best both in terms of accuracy and misclassification cost. Especially, our technique does substantially better for a class that is present in very small proportion and also carries high misclassification penalty. As an aside, we also mention the controversy about test-data quality that we triggered after initial set of results were announced. We conducted a detailed analysis of test-data and proved that the original test-data class labels were wrong.

1.1 Related Work

Various rule-based classification algorithms have been proposed in the literature so far such as CN2[4], the family of AQ algorithms[18], RAMP[2], RISE[7], RIPPER[5], CBA[16], and others[19].

Algorithms CN2, AQ, RIPPER are all sequential covering based techniques (often called separate-and-conquer). CN2 and AQ run into the "small disjuncts" problem exposed extensively in section 1. The RIPPER technique differs slightly from CN2 and AQ algorithms. After discovering (or growing) a highly accurate rule, it immediately prunes it by estimating its generalization error using a separate prune-set. It stops growing the rule-set when the description length of encoding the rule-set and the training data becomes large (MDL principle). Essentially MDL principle allows to balance the generality of rules and their error rate on the training data. RIPPER's intention behind using MDL is that the generality of rules will guard against the small disjunct problems. This seems similar to the remedy based on controlling the specificity of a rule [14] discussed earlier. Techniques such as RAMP[2] combine the general-to-specific and specific-to-general search directions into one single step of learning in an attempt to do annealing-like optimization. RAMP simultaneously strives for minimality of the rule-set and perfect accuracy of the rules on training dataset. Its predictive capabilities are based on keeping rule-set small and general, the underlying hypothesis being the Occam's razor.

Most techniques that grow rules incrementally, differ in their search stopping criterion. Usually the search stops when improvement in some performance metric stops or becomes too small. This performance metric is based on accuracy and coverage of the rule-set on training data, and on cardinality of the rule-set. As with RIPPER and RAMP, most techniques rely on Occam's razor principle which implies that smaller set of general rules generalizes better. The role of Occam's razor in data mining is still being debated [9, 23]. Irrespective of that, one thing remains true - that larger support rules have more generalization capability than smaller support rules. This can be traced back to the early arguments from large-sample theories in statistics. In our PNrule framework, our main emphasis in the rule discovery process is that the rule should satisfy support requirements (along with a reasonable accuracy). We do not specifically strive for small set of rules, but in cases where Occam's razor indeed applies, we believe that PNrule will discover a small set of rules. The other measure that most techniques use is accuracy. The tolerance on accuracy is quite strict in most of the algorithms, except for some algorithms [7] which allow more negative examples to be covered depending on the expected noise in the training data. This is equivalent in some sense to PNrule, which tries to reduce the emphasis on accuracy in favor of support in the first stage. But, our technique ensures that accuracy is regained by removing many false positives in second stage.

Our proposed technique also differs from the decision-tree induction techniques in the following sense. Consider a binary classification problem. We grow all the P-rules to detect presence of the target class. Later we gather *all* the records *collectively* covered by all the P-rules, and learn N-rules on them. Now assume that the decision tree algorithm makes same splitting decisions as our P-rules in the same sequence. This can be achieved by using same rule evaluation metric and restricting each rule to contain similar format of conditions as used by our framework. Now, for low accuracy P-rules, it can be seen that decision-tree algorithm will learn N-rules for *individual* P-rules. It does not have the ability to learn rules on the collection of records covered by all P-rules. This inability, we believe, makes decision-tree algorithms more susceptible to the small disjunct problems, because it has to learn from a smaller set of records. Moreover, our technique has the ability to discover more general N-rules which span across the records covered by different P-rules.

There is another class of techniques [16, 24] that use models based on constrained association rules. These models use all the possible rules present in the training data satisfying some support and accuracy thresholds. This allows them to possibly avoid the local optimas that other techniques might fall into because of their greedy strategy, but proper ways of utilizing all the discovered rules have not yet been established [24].

1.2 Paper Outline

The rest of this paper is organized as follows. We give the details of PNrule framework for binary and multi-class classification problems in Section 2. In Section 3, we present our case study on application of our framework to the KDD-CUP'99 intrusion detection dataset. The paper concludes with a section on research directions.

2 PNrule Classification Framework

PNrule framework is a two-stage process of rule-induction from training data starting with the most general rule, an empty rule. Given a misclassification cost matrix and a training data set with multiple class labels, it learns multiple binary classifier models, one for each class. The model for each class 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. We start this section by illustrating the concept behind our two-stage learning approach. Later, we give detailed algorithms for various steps of the framework.

2.1 Conceptual Illustration of Learning Method

Consider a binary classification problem. Given a training data-set, T, and target class C, a rule is found using the records of T. The rule is of the form $R : A \rightarrow C$, where A is a conjunction of conditions formed by different attributes and their values. Let S denote the subset of T where R applies; i.e. where A is true. R is said to *cover* S. Let S' denote the subset of S where the class label is C. *Support* of the rule is defined as $|S|/|T|$ ($|S|$ denotes the cardinality of set S). *Accuracy* is defined as $|S'|/|S|$.

Given this set of definitions, we will conceptually illustrate our framework using Figure 1. Part (a) shows the entire training data-set, among which the target class is distributed as shown in the shaded region. Our framework operates in two stages. The first stage starts with the entire training set, and finds a rule that has the highest combination of support and accuracy (to be defined later). Let the rule found be indicated by P0. As part (b) of the figure shows, P0 covers a good portion of the shaded area, with very small portion of the unshaded region. Now, we remove the set that is covered by P0, and repeat the process on the remaining set [part (c)]. Let P1 be found on this dataset. P1 still has high support and fairly high accuracy. As the process continues, it becomes increasingly difficult to find rules that have high support as well as high accuracy. In such cases, we give preference to the support as illustrated in part (d), where P2 is preferred over q1 or q2. We stop the process when we are able to capture a sufficiently large portion of the original shaded region [part (a)] or we start running into rules which have very low accuracy. If the accuracy threshold is set higher, then we would stop at P2, else we will cover the remaining positive examples with P3 as in part (e). P3 will be chosen over q3, because despite its lower accuracy, it covers more positive examples as compared to q3. Parts (f) and (g) of the figure show the data-set covered by all the rules combined depending on where the algorithm stops.

As can be seen, because of our preference for support in later iterations, we have covered quite a few examples of the negative class, which are commonly referred to as *false positives.* These are shown in shaded areas of (f) and (g).

Now, on the dataset consisting of records covered by the union of all P-rules, we start an inverse learning process. The goal here is to learn rules that will cover (i.e. *remove*) most of the false positives *collectively* covered by the P-rules. We follow similar steps as described above, but our new target class is now the *absence of original* target class. In Figure 1(f), this is shown by a restricted universe with shaded portion as the new target class. Again first rule N0 tries to capture as much of the positive examples of the new target class with high accuracy. As iterations progress, we still give preference to the support of rules, while closely monitoring the accuracy. The point to note is that a 100% accurate rule in this stage strictly removes the false positives covered by the first stage, while a rule with less than 100% accuracy removes some of the true positive examples of the original target class (that were captured in the first stage). We call this phenomenon as *introduction of false negatives.* Our goal in this stage is to remove as many false positives as possible, while not introducing too many false negatives. Choosing rules which have high combination of support and accuracy, helps us achieve this goal. All the rules discovered during this stage are called N-rules.

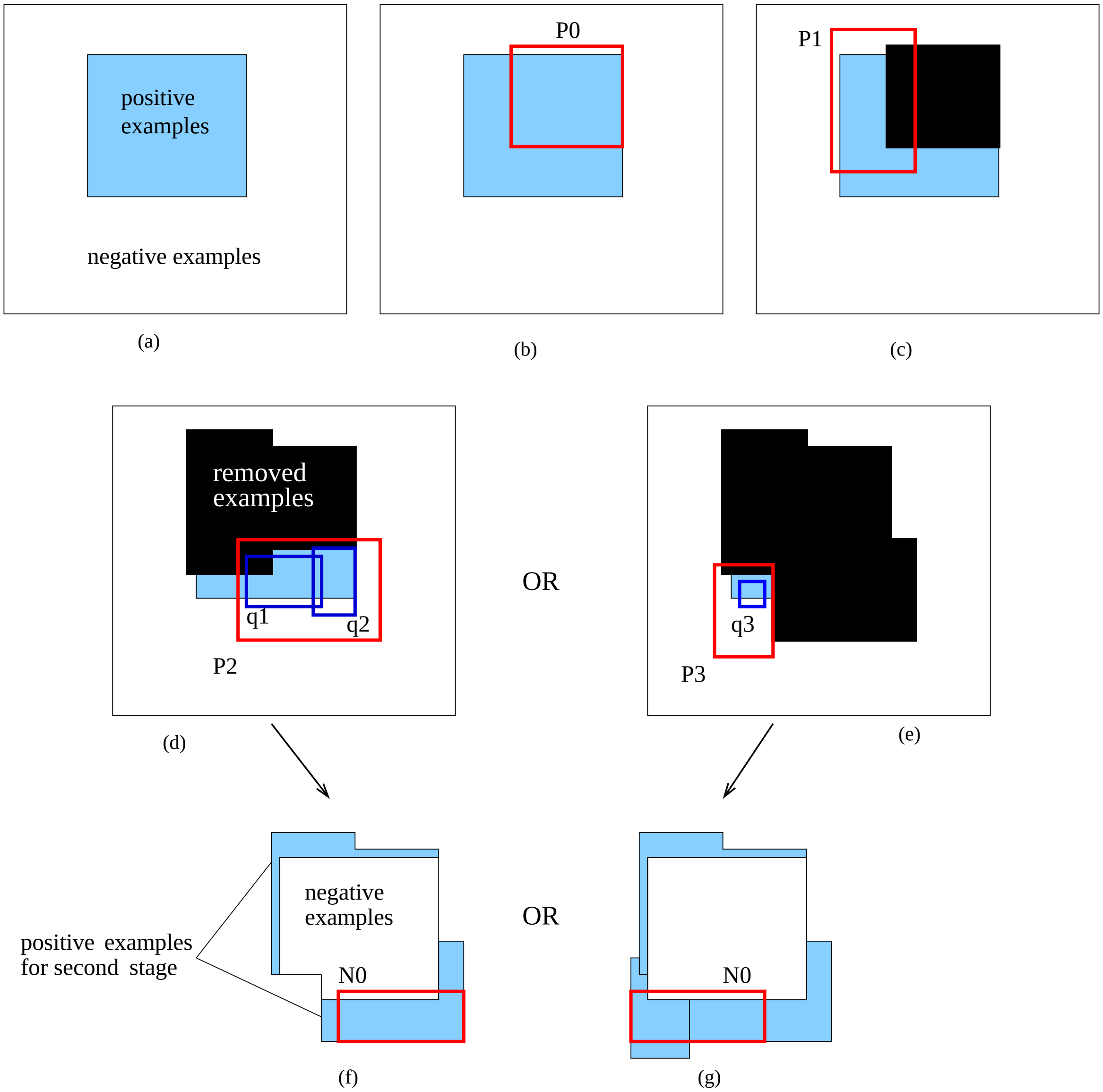


Figure 1: How PNrule works. (a) Original training set, (b) Discover first P-rule, (c) Discover Second P-rule on remaining examples (d) Choice of Third P-rule. P2 chosen over q1 or q2, because of its support. (e) If accuracy threshold is low, P3 will be selected. Again, P3 is chosen over q3 because of its support. (f) Starting data-set for second stage if first stage stops after P2. (g) Starting data-set for second stage if first stage stops after P3.

During each of the stages, higher accuracy large support rules are discovered in the beginning, and lower accuracy rules are discovered towards the end. We rank the rules in the order they are discovered, the rules at the beginning being more significant than the rules towards the end.

At the end of this two-stage process, we expect to have captured most of the positive examples of the target class, with few of the negative examples (false positives).

Most of the false positives still getting covered can be attributed to the lower accuracy P-rules. Similarly, most of the positive examples missing from the coverage can be attributed to the lower accuracy N-rules. Based on this observation, we design a scoring mechanism that allows to recover some of the false negatives introduced by the low ranked N-rules. Also, the scoring mechanism will try to assign low scores to the negative examples covered by low accuracy P-rules. Note that we can afford to be more aggressive by keeping the final accuracy threshold low in each of the stages, because we rely on our scoring mechanism to correct for the additional errors introduced.

The *two-stage* learning approach illustrated above and the scoring mechanism, elaborated in Section 2.4 later, are the two key novel features of our method.

2.2 Main Learning Algorithm and Model Format

The pseudo-code of the main algorithm for learning a binary classifier model is given in Figure 2. The details of subroutines ChooseBestRule and ComputeScores are given in following subsections. The points to note from the main algorithm are the following.

- ▷ The algorithm is parametrized by the support and accuracy thresholds applied to each stage. From our experience with the case-study problem, which had wide variation of class distributions, usually the support thresholds in both stages are quite strict (higher). The accuracy thresholds can be relaxed (lowered) depending on the characteristics of the target class. Especially for smaller target classes, they might need to be lowered substantially for the P-stage, if the support thresholds are to be set higher.
- ▷ If the scoring mechanism is absent, then our model 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. 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

$$\begin{aligned} C = \; & (P_0 \wedge \neg N_0 \wedge \neg N_1 \wedge ... \wedge \neg N_{n_N-1}) \vee \\ & (P_1 \wedge \neg N_0 \wedge \neg N_1 \wedge ... \wedge \neg N_{n_N-1}) \vee ... \vee \\ & (P_{n_P-1} \wedge \neg N_0 \wedge \neg N_1 \wedge ... \wedge \neg N_{n_N-1}) \end{aligned}$$

  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 we can learn using our model. However, as we will see in section 2.4, our scoring mechanism allows to relax this restriction, by selectively deciding to ignore the effects of certain N_j rules for a given P_i.

2.3 Choosing and Evaluating Rules

The steps of the ChooseBestRule subroutine called from main algorithm are given in Figure 3. It can be seen from the algorithm that each rule is quite general in the sense that it involves no conjunctions. The rules for categorical attributes are straightforward. For numerical attributes, mechanism to find the interesting ranges of values is crucial in the formation of rules. Currently, we use a simple clustering mechanism. We first start by forming a small number of ranges of equal span. We merge or split the ranges such that the number of records in each range satisfy certain pre-specified minimum and maximum requirements on the cluster size. After this, we evaluate the strength of each range, and merge the adjacent ranges which have similar strengths. A more sophisticated algorithm would be to use this simple algorithm to produce promising ranges. Each of these promising ranges can then be systematically extended or reduced so as to maximize the performance metric used for the rule.

We define two performance metrics that are used for evaluating and comparing the rules. The attempt here is to capture distinguishing capability of the rule for a given class, the support of the rule, and accuracy

```
C: Target Class
T: Training Set with 1 or 0 as class (1 when class = C, 0 otherwise)

LearnPNruleModel( T, C, MinAccuracyP, MinAccuracyN, MinSupportP, MinSupportN, MinSupportScore, MinZ )

    —- First Stage (P-Stage): Discover P-rules
    S = T
    TargetClass = C
    $n_P = 0$
    Nseed = empty
    do
        $P_{n_P}$ = ChooseBestRule( S, TargetClass )
        Q = records of S covered by $P_{n_P}$
        Nseed = Nseed U Q
        S = S − Q
        $n_P = n_P + 1$
    while( Accuracy($P_{n_P - 1}$) > MinAccuracyP && Support($P_{n_P - 1}$) > MinSupportP )

    —- Second Stage (N-Stage): Discover N-rules
    S = Nseed
    TargetClass = not C
    $n_N = 0$
    do
        $N_{n_N}$ = ChooseBestRule( S, TargetClass )
        Q = records of S covered by $N_{n_N}$
        S = S − Q
        $n_N = n_N + 1$
    while( Accuracy($N_{n_N - 1}$) > MinAccuracyN && Support($N_{n_N - 1}$) > MinSupportN )

    —- Gather PNrule statistics
    for( $i = 0$ ; $i < n_P$ ; $i++$ )
        for( $j = 0$ ; $j < n_N$ ; $j++$ )
            SupportMatrix( $i$ , $j$ ) = number of records in T where both $P_i$ and $N_j$ apply
            ErrorMatrix( $i$ , $j$ ) = number of records in T where Both $P_i$ and $N_j$ apply, but the class is 1
        end for
        SupportMatrix( $i$ , $n_N$ ) = number of records in T where $P_i$ applies and no N-rule applies
        ErrorMatrix( $i$ , $n_N$ ) = number of records in T where $P_i$ applies and no N-rule applies,
                but the class is 0
    end for
    ScoreMatrix = ComputeScores( SupportMatrix, ErrorMatrix, MinSupportScore, MinZ )

    Output P-rules $P_i$ $(0 \leq i < n_P)$, N-rules $N_j$ $(0 \leq j < n_N)$, and ScoreMatrix
end
```

Figure 2: Algorithm to Learn PNrule Model for Binary Classification

```
ChooseBestRule( S, C )
    RS = empty;
    for each Attribute-type pair (A,type)
        if type is categorical
            for each distinct value, v, of A in S,
                Form rules, R1: (A = v) → C and R2: (A != v) → C
                Compute strengths of R1 and R2 on S w.r.t. C
                Add R1 and R2 to RS along with their strengths
            endfor
        endif
        if type is continuous
            Form Interesting Ranges of A's Values using S.
            for each range [low,high)
                Form rules, R1: ( A in [low,high) ) → C and R2: ( A not in [low,high) ) → C
                Compute strengths of R1 and R2 on S w.r.t. C
                Add R1 and R2 to RS along with their strengths
            end for
        endif
    endfor
    sort rules in RS in increasing order of their strength
    return the rule R having highest strength
end
```

Figure 3: Algorithm to Choose Best Rules in Each Iteration

of the rule, all in one single metric. A rule with higher value of the metric should imply that it is statistically more significant in capturing the target class.

- **Z-number:**
  Let a_R denote the accuracy of a given rule, R, and s_R denote its support. Refer to the beginning of section 2.1 for definitions. Let a_C denote the mean of target class C, defined as $a_C = |S_C|/|S|$, where S is the current training data set, and S_c is the subset of S where C is true. Let σ_C denote the standard deviation of target class C. For the binary problem under consideration, $\sigma_C = \sqrt{a_C(1 - a_C)}$. Using these notations, Z-number is defined as

  $$Z_R = \sqrt{s_R}\ (a_R - a_C)/\sigma_C$$

  This metric measures how many standard deviations away the mean of the rule is from the mean of the target class. The farther away a_R is from a_C, better can R distinguish examples of class C. Weighing this *distance* by s_R gives more weight to the high support rules. Z-number is similar to the z-test or t-test from the statistics, depending on the value of s_R. A rule with high positive Z-number ($a_R \gg a_C$) predicts presence of C with high confidence. Similarly, a rule with high negative Z-number ($a_R \ll a_C$) predicts absence of C with high confidence. This metric is similar to the Z-test used in statistics.

- **Y-number:**
  When the class mean, a_C is closer to 1; i.e. when we are discovering rules for large classes, Z-number defined above is not capable of distinguishing well between high accuracy rules. It tends to give a little too much weight to the support. In such cases, we define a Y-measure which computes the ratio of how far away a_C and a_R are from ideal mean of 1.0 and weighs this ratio with support. The goal is to assign more weight to the high accuracy rules.

  $$Y_R = \sqrt{s_R}\ min(\sqrt{s_R},\ (1.0 - a_C)/(1.0 - a_R))$$

  To illustrate the necessity of Y-number, consider an example where $a_C = 0.95$. A rule R1 has $a_{R1} = 0.99$ and $s_{R1} = 10000$, whereas another rule has $a_{R2} = 0.98$ and $s_{R2} = 20000$. Then $Z_{R1} = 18.3532$ and

$Z_{R2} = 19.4666$. Whereas their Y-numbers are $Y_{R1} = 500$ and $Y_{R2} = 353.55$. Based on Z-number we would have given preference to R2, but it has lower accuracy rule as compared to R1, despite their supports being of the same order. Y-number will choose the desired rule, R1. Note that this situation is more likely to happen in the first few iterations of any stage, where PNrule strives for high accuracy rules just like other rule-induction techniques do. Conversely, for small a_C values (either target class is small to begin with or in the later iterations), Y-number starts getting biased excessively more towards high accuracy, low support rules because of larger $1/(1 - a_R)$ term. For example, for $a_C = 0.75$, if R1 has $a_{R1} = 0.98$ and $s_{R1} = 300$, and R2 has $a_{R2} = 0.90$ and $s_{R2} = 5000$, we get $Z_{R1} = 4.62$, $Z_{R2} = 16.33$, $Y_{R1} = 216.51$, and $Y_{R2} = 176.78$. Here, clearly R2 has more statistical support compared to R1, reflected accurately in Z-number. Hence, for low a_C values, we revert back to Z-number.

2.4 PNrule Classification Strategy and Scoring Algorithm

Once we have learned P-rules and N-rules for each class, first we describe how we use them to classify an unseen record.

As indicated in section 2.1, P-rules and N-rules are arranged in decreasing order of significance, which is same as their order of discovery. Given a record consisting of attribute-value pairs, each classifier first applies its P-rules 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. The reason for having the last default N-rule will become clear little later in this section. If our 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, this is not useful, especially in the multi-class framework, where we may need to resolve conflicts between True decisions of multiple classifiers. We need a mechanism to assign a *score* to each decision. 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%). This score can be interpreted as the probability of the given record belonging to the target class. Scores from individual classifiers are combined with the cost matrix to decide the most cost-effective class for the given record. This is the overall classification strategy.

In the light of this, we now describe how each classifier determines the scores to assign to each P-rule, N-rule combination.

The motivation behind the design of scoring mechanism 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* covered 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.

The starting point of the scoring mechanism are the SupportMatrix and ErrorMatrix defined in Figure 2. In SupportMatrix, entry (i,j) $[j < n_N]$ gives the number of records for which the true predictions made by P-rule P_i were converted to false by N-rule N_j. Last entry in row i, SupportMatrix(i,n_N) gives the number of records where P_i applied but no N-rule applied. The ErrorMatrix reflects the prediction errors made by each (P_i,N_j) combination. Entries ErrorMatrix(i,j) $[j < n_N]$ give false negatives introduced by N_j for P_i's predictions, whereas ErrorMatrix(i,n_N) gives the number of false positives of P_i that none of the N-rules was able to remove. The last column effectively corresponds to a rule which states "no N-rule applies". An example of SupportMatrix and ErrorMatrix is shown in Figure 5. Look at the entries for [P1,N1] in both matrices. They imply that among the records of training dataset covered by rule P1, rule N1 applied to 7 records (SupportMatrix[P1,N1]), out of which its decision to remove false positives was wrong for 2 records (ErrorMatrix[P1,N1]). This means that it removed 5 false positives of P1, and introduced 2 false negatives for P1.

Using the SupportMatrix and ErrorMatrix, our goal is to come up with a ScoreMatrix, such that ScoreMatrix(i,j) $(j < n_N)$ gives a score to the record for which both P-rule P_i and N-rule N_j apply, and ScoreMatrix(i,n_N) gives a score when P-rule P_i applies and no N-rule applies. We use the algorithm

```
ComputeScores( SupportMatrix, ErrorMatrix, MinSupport, MinZ )
    for( $i = 0$ ; $i < n_P$ ; $i + +$ )
        TruePositiveVariation( $i$, $n_N$ ) = SupportMatrix( $i$, $n_N$ ) - ErrorMatrix( $i$, $n_N$ )
        FalsePositiveVariation( $i$, $n_N$ ) = ErrorMatrix( $i$, $n_N$ )
        AccuracyVariation( $i$, $n_N$ ) = TruePositiveVariation( $i$, $n_N$ ) /
                (FalsePositiveVariation( $i$, $n_N$ ) + TruePositiveVariation( $i$, $n_N$ ))
        for( $j = n_N - 1$ ; $j \geq 0$ ; $j - -$ )
            TruePositiveVariation( $i$, $j$ ) = TruePositiveVariation( $i$, $j + 1$ ) + ErrorMatrix( $i$, $j$ )
            FalsePositiveVariation( $i$, $j$ ) = FalsePositiveVariation( $i$, $j + 1$ ) + SupportMatrix( $i$, $j$ ) $-$
                    ErrorMatrix( $i$, $j$ )
            AccuracyVariation( $i$, $j$ ) = TruePositiveVariation( $i$, $j$ ) /
                    (FalsePositiveVariation( $i$, $j$ ) + TruePositiveVariation( $i$, $j$ ))
        endfor
    endfor
    for( $i = 0$ ; $i < n_P$ ; $i + +$ )
        for( $j = 0$ ; $j < n_N$ ; $j + +$ )
            parentSupport = TruePositiveVariation( $i$, $j$ ) + FalsePositiveVariation( $i$, $j$ )
            if( parentSupport < 2 * MinSupport )
                Assign AccuracyVariation( $i$, $j - 1$ ) to ScoreMatrix( $i$, $j..n_N$) and go to next $i$
            leftTP = TruePositiveVariation( $i$, $j$ ) - TruePositiveVariation( $i$, $j + 1$ )
            leftFP = FalsePositiveVariation( $i$, $j$ ) - FalsePositiveVariation( $i$, $j + 1$ )
            leftZ = Z-number of left node w.r.t. Parent's distribution
            if( leftTP + leftFP > MinSupport && |leftZ| > MinZ )
                ScoreMatrix( $i$, $j$ ) = leftTP / (leftTP + leftFP) $^*$
            else
                ScoreMatrix( $i$, $j$ ) = AccuracyVariation( $i$, $j$ ) $^*$
            endif
        endfor
        ScoreMatrix( $i$, $n_N$ ) = AccuracyVariation( $i$, $n_N$ ) $^*$
    endfor
    return ScoreMatrix
end
```

Figure 4: Algorithm for Constructing a Mechanism to Assign Scores to Decisions (*see section 2.4 for some details)

given in Figure 4 to arrive at this ScoreMatrix.

Let us illustrate the idea behind the algorithm using the example given in Figure 5. We first construct the matrices TruePositiveVariation, FalsePositiveVariation, and AccuracyMatrix. A P-rule captures some positive examples (True Positives, or TP) and a few negative examples (False Positives, or FP), when it is discovered first. These together give it its *initial* accuracy, TP/(TP+FP). Look at the Init columns in the Figure. As N-rules are applied successively, the accuracy varies depending on how many false positives are removed and how many false negatives are introduced by each N-rule. Precisely these variations are reflected in the three matrices. The matrices can be understood better via a decision tree for each P-rule. Figure shows such a tree for rule P1. The root node A has all the records where P1 applies. There are 65 such records for P1, out of which 53 are TPs and 12 are FPs (accuracy of 81.5%). Out of these records, first N-rule N0 applies to 3 records (1 TP, 2 FP). Now, we determine the significance of N0 specific to P1, by applying our first criterion, which states that *support of any decision should satisfy a MinSupportScore threshold*. For our example, threshold is 5, hence N0 has statistically insignificant support, and we decide to ignore its effect on P1. The decision is reflected in the ScoreMatrix by assigning the accuracy of the parent node to the [P1,N0] location (81.5%). Now, we recalculate the TP, FP, and Accuracy statistics for the records where N0 did not apply. We cannot propagate the statistics of root node to node B, even though we decide to ignore N0's effect on P1. The reason is the sequential covering nature of the way N-rules are learned, which implies that the decisions made by rule N1 (and later rules) are significant only to the population of records where rule N0 does not apply.

Now, when N1 is applied to the new set of records (52 TP, 10 FP), it applies to 7 of those (2 TP, 5 FP). It satisfies our support criterion of significance ($\geq$ MinSupportScore). Now, we calculate the Z-number of N1 w.r.t P1, given by formula $Z_N = \sqrt{n_P}(a_N - a_P)/\sigma_P$, where n_P is the support of parent node (TP+FP). a_N and a_P are accuracies of N-rule's node and parent, respectively, and $\sigma_P = \sqrt{(a_P)(1-a_P)}$ is the standard deviation of parent's population. Our second criterion of significance states that *if the absolute value of Z_N is sufficiently high ($\geq$ MinZ), then the decision made by the N-rule is significant w.r.t. the given P-rule.* This test is similar to the z-test or t-test from statistics (depending on the value of n_P). Point to note here is that each N-rule had a significant Z-number when it was discovered in the learning process because it was computed over a collection of records covered by all P-rules. What we are determining here is its significance specific to a given P-rule. In our example, P1-specific |Z| value of N1 is high (11.85 $\geq$ MinZ=3.0), so we decide that N1's effect on P1 is significant. The decision is reflected in the ScoreMatrix by assigning the accuracy of N1's node to the [P1,N1] location (28.6%). So, whenever N1 applies to a record predicted true by P1, we say that the probability of of that record belonging to the target class is only 28.6%.

The process continues for N2, where we find that N2's decision has significant support, but it does not have sufficient distinguishing capability w.r.t P1 (low |Z|). Hence, we ignore its effect on P1, and assign the ScoreMatrix[P1,N2] location the accuracy of N2's parent (90.9%). Finally, when no N-rule applies, we assign the accuracy of N3's leaf to the last location in P1's row (92.0%). This entire process is repeated for P0 and P2. At every node of the decision tree, we determine whether a N-rule is significant w.r.t. to the given P-rule. If it is significant, we use the accuracy of the N-rule to score the decision, or else we use the accuracy of its parent. It can be verified that for P0, none of the N-rules have statistically significant support, and for P2, every N-rule is significant (support as well as the Z criterion).

Here are some more points to note about the algorithm, which are not illustrated by the above example. First of all, if any node's support falls below MinSupportScore, we ignore its effect, and assign it the score of its nearest ancestor having statistically significant support. Second, we do not allow a perfect decision at any node; i.e. our scores are never exact 100% or 0%. A score of 100% gets adjusted to $n/(n+1)$ where $n = TP$, whereas a score of 0% gets adjusted to $1/(n+1)$, where $n = FP$. The reason for doing this is to give less importance to the perfect decisions made on small population as compared to the perfect decisions made on larger population. Also, the moment a parent node is found to be perfect (before adjusting scores), we stop splitting it further and assign its adjusted score to all the remaining locations of its row. Finally, the parameters MinSupportScore and MinZ can usually be fixed for most problems using statistical arguments.

The essential effect of this 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 adjusted probability that a record belongs to the target class, if P_i, N_j combination applied to it.

SupportMatrix

	N0	N1	N2	N3
P0	0	0	4	100
P1	3	7	5	50
P2	8	5	6	27

ErrorMatrix

	N0	N1	N2	N3
P0	0	0	3	1
P1	1	2	4	4
P2	0	1	2	4

TruePositiveVariation

	Init	N0	N1	N2
P0	102	102	102	99
P1	53	52	50	46
P2	26	26	25	23

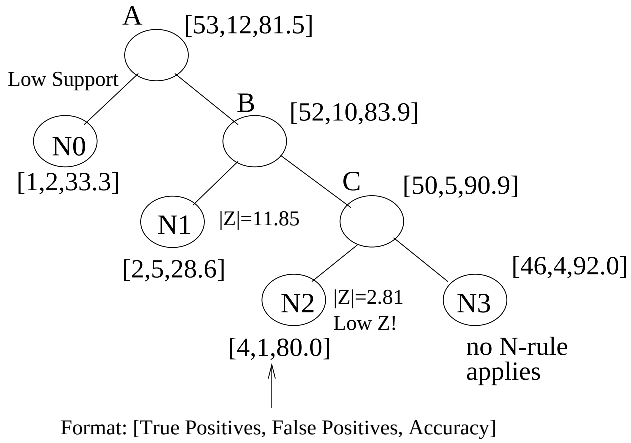
FalsePositiveVariation

	Init	N0	N1	N2
P0	2	2	2	1
P1	12	10	5	4
P2	20	12	8	4

AccuracyVariation

	Init	N0	N1	N2
P0	98.1	98.1	98.1	99.0
P1	81.5	83.9	90.9	92.0
P2	56.5	68.4	75.8	85.2

Illustration for P-rule P1:



Parameters:

MinSupportScore = 5
MinZ = 3.0

Final Result: ScoreMatrix

	N0	N1	N2	N3
P0	98.1	98.1	98.1	99.0
P1	81.5	28.6	90.9	92.0
P2	11.1	20.0	33.3	85.2

Figure 5: Illustration of Constructing the Scoring Mechanism (ScoreMatrix)

2.5 Making PNrule Cost-sensitive

Given a misclassification cost matrix $\{C(s,t)\}$, where $C(s,t)$ is the cost of predicting class s as class t, the goal is to predict the classes of a given data set to minimize the total misclassification cost penalty.

Given a record x, if the actual probability $P(s|x)$ of the record belonging to class s is known, then Bayes optimality rule [10] implies that assigning a class t to the record that minimizes $\sum_s P(s|x)C(s,t)$ gives the least overall cost. We use the scores generated by our binary classifiers as the estimation of $P(s|x)$, and use this formula to predict the class of x.

This strategy may not work well if the scores generated by our classifier are not close estimates of $P(s|x)$. We have not analyzed this issue in detail, but plan to do so in the future. But, from preliminary concept behind our scoring strategy, it can be seen that if test-data and training-data have similar class distributions, then our scores will be closer to true estimates. In section 4, we discuss some possible ways of handling situations where our scores are not close to true estimates.

3 Case Study: Applying PNrule to Detect Network Intrusions

In order to validate our PNrule framework, we applied it to a classification problem from the domain of network intrusion detection. The training and test data-sets were supplied as part of KDD-CUP'99 Contest[11]. We participated in the contest. In this section, we explain the problem and its challenges. We describe the strategy that we used initially. We analyze the strategy critically to give reasons as to why it did not do so well, but before that we also explain how we suspected the original test-data for its quality, and how we proved it wrong. The PNrule framework proposed in this paper is an improved and automated version of that original strategy. We finally show that PNrule approach is promising, as it yields the best performance under certain scenarios.

3.1 The Data-Set and Challenges

The network intrusion detection problem provided as part of the KDD-CUP-99 classifier learning contest [11] was as follows. Given a training data-set of close to 5 million records belonging to five classes and a misclassification cost matrix, goal was to learn a classifier model so as to achieve least total misclassification cost of predicting the labels of the supplied test-data records. Here is the description of the data-set:

- ▹ The training and test-data were collected from a real-life scenario of a military computer network which was intentionally peppered with various attacks that hackers would use to break in.
- ▹ Each record represents a connection between two network hosts. It is characterized by 41 attributes, 34 continuous-valued and 7 discrete-valued. Some exemplary attributes are duration-of-connection, number-of-bytes-transferred, number-of-failed-login-attempts, network-service-to-which-connection-was-made, etc.
- ▹ Each record represented either an intrusion (or attack) or a normal connection. There are four categories of attack: denial-of-service (dos), surveillance (probe), remote-to-local (r2l), and user-to-root (u2r).

As can be seen, this data-set was quite large and it represented a real-world problem. Here are some salient features of the problem and data-set that made this contest challenging:

- ▹ The goal was not mere accuracy, but misclassification cost matrix. This cost matrix is given in Table 1(a).
- ▹ Each attack category had some subclasses of attacks. We were told that out of total 39 total attack subclasses that appear in test-data, only 22 were present in the training data.
- ▹ The distribution of training records among attack categories as well as subclasses varied dramatically. Table 1(b) shows the counts for some of the representative classes and subclasses. Moreover, the misclassification cost penalty was the most for one of the most infrequent classes, r2l.
- ▹ It was told that the test-data had a completely different distribution of classes as compared to the training-data.

		predicted class				
		normal	probe	dos	u2r	r2l
actual class	normal	0	1	2	2	2
	probe	1	0	2	2	2
	dos	2	1	0	2	2
	u2r	3	2	2	0	2
	r2l	4	2	2	2	0

(a)

Class	Count
normal	972781 (19.9%)
dos	2883370 (79.3%)
probe	41102 (0.84%)
r2l	1126 (0.023%)
u2r	52 (0.001%)

Subclasses	Count
smurf (dos)	2807886
neptune (dos)	1072017
back (dos)	2203
teardrop (dos)	979
ipsweep (probe)	12481
satan (probe)	15892
warezclient (r2l)	1020
buffer_overflow (u2r)	30

(b)

Table 1: Characteristics of Problem and Training Data. (a) The misclassification cost matrix. (b) Class and subclass distribution in training data.

▷ We had only 25 days to submit results. And the evaluation criterion based on misclassification costs was announced only two weeks prior to the submission date.

3.2 Our Original Strategy: Results and Critique

This is the method we used for submitting our results to the contest. We describe it here because it has the roots of PNrule framework in it. We developed it in a three week period starting from scratch, without using any existing classifiers. Here is the summary of our strategy.

1. We decided not to use any domain knowledge. We decided to evolve a rule-based model which has rules with high support and good accuracy in training data. The underlying ”bias” we assumed was that such rules would yield the model better generalization capability.

2. We observed a peculiar behavior of the class labels in training data. Most of the labels were appearing in *bursts*; i.e. large number of consecutive records had the same class label. In fact, 4,898,431 records of training data appeared in only about 620 bursts.

3. Since the classes were distributed widely, we decided to learn a separate binary model for each class, starting with the most frequent classes smurf, neptune, and normal. We observed that for smurf, one single rule was capturing all the positive examples. It had a very high accuracy of 99.87%. When we applied a burst analysis to this to remove records that did not occur in large bursts, we could bring the accuracy up to 99.999%. We could also find a P- and N-rule based model for neptune with sufficiently high accuracy in training data, 99.98%.

4. We could not find such high accuracy rules for any other class, hence we decided to remove all the smurf and neptune records and learn the models for other classes on the remainder of the data. This was done to increase the relative proportion of really tiny categories such as r2l and u2r.

5. For every class other than smurf, we could not find a set of rules that together could capture large portion of positive examples with high accuracy. Hence, we formed rule-based model in two stages similar to PNrule. We first found all rules based on single attribute in the manner described in section 2.3. We ranked them based on Z-number. A few rules among those with very high positive Z-number were hand-picked to cover the positive examples of the class. A few N-rules were similarly hand-picked from the rules having very low negative Z-numbers, since such rules predict absence of the target class with high confidence. For each P-rule, we observed how many of its false positives were getting removed by some N-rule. For each N-rule, we observed how many false positives it is removing and how many false negatives it is introducing. Using these measures, we went through a few iterations of adding, deleting, and reordering of P-rules and N-rules, to arrive at a final set which had sufficiently low false positive rate and as high detection rate as we could achieve. The decision made using this model was as follows: If some P-rule applies and none of the N-rules applies, we predict the class to be true, or else it was predicted false.

	normal	probe	dos	u2r	r2l	Acc
normal	59958	534	163078	2	22	26.8%
probe	1968	2191	7	0	0	52.6%
dos	6775	23	60054	0	0	89.8%
u2r	197	0	23	7	1	3.1%
r2l	14759	11	3	2	1414	8.7%
FP-rate	29.3%	21.6%	73.1%	36.4%	1.7%	
Misclassification Cost = 402000, Accuracy = 39.75%						

	normal	probe	dos	u2r	r2l	Acc
normal	59958	534	**77**	2	22	**99.0%**
probe	1968	2191	7	0	0	52.6%
dos	6775	23	**223055**	0	0	**97.0%**
u2r	197	0	23	7	1	3.1%
r2l	14759	11	3	2	1414	8.7%
FP-rate	29.3%	21.6%	**0.05%**	36.4%	1.7%	
Misclassification Cost = 75998, Accuracy = 92.15%						

Figure 6: Results obtained with original two-stage strategy. (a) With corrupt test-data supplied initially. We proved this test-data wrong. (b) With correct test-data.

	DataSet1 (DS1)	DataSet2 (DS2)	DataSet3 (DS3)	DataSet4 (DS4)
Description	Normal in Training	Normal in Test Except those in DS4	Smurf in Training	Disputed Records in Test
Counts	972,781	60,590	2,807,886	164,096
Labels in Disputed Records				normal: 163,004 smurf: 1,090
Our Labels in Disputed Records				smurf: 164,096
#times our simple smurf rule applies	3456 (0.35%)	173 (0.28%)	2,807,886 (100%)	164,091 (99.99%)
#times our strong smurf model applies	19 (0.002%)	0 (0%)	2,805,850 (99.93%)	163,582 (99.69%)
# distinct values for Atr0 (duration)	9034	224	1	1
Max value for Atr0	58,329	54,451	0	0
# distinct values for Atr4 (src_bytes)	7,145	2,354	2	8
Max value for Atr4	89,581,520	6,291,668	1,032	1,032

Table 2: How we Proved the Original Test-Data labels wrong.

6. After forming the models, we applied burst analysis to remove bursts having small size and lower accuracy. This appeared to work very well for training data, and also the predictions of our rules in test-data seemed to exhibit bursty behavior, especially for large classes (smurf, etc).

7. We finally brought in costs in an ad-hoc manner. Whenever there was a conflict in decisions, we decided to give preference to the classes in order of their misclassification cost penalties.

When the results came out, we had 8th rank among 24 contenstants, and we had the confusion matrix as shown in Figure 6(a). When the test-data labels were made available, we observed the following:

- Our assumption about similar "bursty" behavior in training and test data was wrong for most of the classes. If we would not have done the analysis, and would have used the predictions made by our models directly, we would have improved our misclassification cost to 401008, and our rank could have gone up to 5th place.
- The first thing that struck us in the confusion matrix was that our false alarm rate for dos was very high. This led us to trigger a controversy about test-data quality issue, which we describe in next subsection.

3.2.1 Test-Data Quality Issue: How we proved it wrong

Our very high false alarm rate for dos was quite surprising, given the very high accuracy models we had found for smurf and neptune (two prominent subclasses of dos). In fact, we were quite accurate in predicting neptune records in test-data, but apparently almost all of the 164,096 records we had predicted to be smurf were normal according to the test data labels. Our smurf model had a low false positive rate of 0.35%, but was based on a rule with only one attribute. So, we added more conjunctions to the rule, and made it consist of 31 attributes out of 41. With this stronger model, the false positive rate had gone down to 0.002%, and we could still capture 99.93% of smurf in training. So, what are the statistical chances of a 0.002% false positive rate blowing all the way upto 99.3% in test-data? If they are indeed very high, then it would make almost every data-mining technique to fail.

Then, we did some more analysis. We tried to use domain knowledge. We observed the behavior of three basic attributes: the duration of a connection, and bytes transferred from and to the source host. For a *normal* connection, these attributes should vary all over their possible range of values, whereas for *attack* connections, they should exhibit some standard pattern based on some hackers strategy of attack. Hence we separated four data-sets, and observed the behavior of these basic attributes in them. The definition of data-sets and results are shown in Table 2. As can be seen, The first two datasets (DS1 and DS2) are very similar, whereas the last two datasets (DS3 and DS4) are very similar, making a case that most records in DS4 (disputed data-set) should be smurf rather than normal.

Using this argument along with our statistically stronger model, we made our case that the test-data labels, especially those in the disputed data-set, were wrong. And, indeed we were right. Our proof reached just in time to hold the results announcements, and the people who had prepared the data agreed that they had made a mistake in labeling the test-data. The new corrected test-data was supplied, and all contestants were re-evaluated using it. Our rank had improved two notches, up to 6th, with this new test-data. The new confusion matrix is shown in Figure 6(b). As can be seen, our false alarm rate for dos is almost close to 0.0%. In fact, all the labels in the disputed data-set were smurf.

3.3 Applying PNrule and Results

The original strategy we used had two main shortcomings. First, the rule-based models were formed using hand-picked P- and N-rules. It took us a few iterations of choosing good rules that give low false positive rate as well as high detection rate. Second, the decision made by each classifier model was binary, 0 or 1. So, all N-rules had equally strong influence on each P-rule. Also, a binary decision has a score of either 0% or 100%, which means putting too much confidence in the decision of each classifier.

The PNrule framework of section 2, was developed after noticing these shortcomings. Main deviations in PNrule framework from the pre-PNrule strategy are the systematic automated way of finding P-rules and N-rules using sequential covering strategy, and the scoring mechanism. The usual criticism of sequential covering based algorithms, regarding small disjunct problem, was elaborated upon in section 1.

Apart from these differences in the algorithm, we did following things differently while applying PNrule to the network intrusion detection data-set of KDD-CUP'99 contest:

1. First obvious thing we did was not to rely on any burst analysis. This required learning a 2-stage PNrule model for smurf also, in order to remove the false positives of the P-rule.

2. We first developed models for smurf and neptune using the entire training set T, but now instead of removing every record where smurf and neptune were predicted true, we removed only those records which had a score greater than 99.9% for these classes. We refer to the filtered training data-set as $T1$.

3. Two prominent classes remaining were normal and probe. The other remaining classes, r2l, u2r, and remaining subclasses of dos, were really tiny. We formed a 10% subset of $T1$. This subset, refered to as $T1_{10\%}$, had every record belonging to these classes, but only around 10% sample of the records belonging to normal and probe. The goal was to increase the statistical significance of the tinier classes. We learned P-rules for normal and probe using entire $T1$. But, we learned N-rules for normal and probe, and entire models (P- and N-rules) for other smaller classes using $T1_{10\%}$.

4. We used scores of each of the classifiers along with the misclassification cost matrix, to make final decisions according to the procedure given in section 2.5.

Before describing the overall results, let us illustrate how the scoring mechanism works. The model formed for r2l consists of 5 P-rules and 15 N-rules. Scores for individual PN rule combinations are formed using the $T1_{10\%}$ data-set. The matrices formed during the ComputeScores algorithm are shown in Figure 7. The parameters used are MinSupport=10 and MinZ=5.0. The variations of true and false positives illustrate how N-rules work to improve the accuracy by removing false positives. All locations of ScoreMatrix in the

TruePositiveVariation

	Init	N0	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	N11	N12	N13	N14
P0	274	274	274	274	274	274	274	274	274	274	264	264	264	264	254	254
P1	733	710	710	710	709	703	698	698	697	689	689	687	663	642	640	639
P2	51	51	51	51	51	45	44	44	44	44	44	44	34	32	32	31
P3	12	12	12	12	12	10	10	10	10	10	10	10	9	8	8	4
P4	39	39	39	39	39	39	38	38	38	38	38	38	38	34	19	19

FalsePositiveVariation

	Init	N0	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	N11	N12	N13	N14
P0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P1	3818	1496	918	885	652	370	276	276	276	209	167	130	60	59	29	21
P2	6	6	6	6	4	4	4	4	2	2	2	2	2	1	1	1
P3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
P4	380	380	378	291	256	245	242	138	59	59	44	41	38	2	2	2

ScoreMatrix

	N0	N1	N2	N3	N4	N5	N6	N7	N8	N9	N10	N11	N12	N13	N14	N15
P0	99.6	99.6	99.6	99.6	99.6	99.6	99.6	99.6	99.6	99.6	99.6	99.6	99.6	99.6	99.6	99.6
P1	0.9	0.1	3.0	4.3	2.1	5.1	71.7	71.7	10.7	2.4	5.1	25.5	91.7	6.3	95.7	96.8
P2	89.5	89.5	89.5	89.5	92.7	91.8	91.7	91.7	95.7	95.7	95.7	95.7	94.4	97.0	97.0	96.9
P3	92.3	92.3	92.3	92.3	92.3	92.3	92.3	92.3	92.3	92.3	92.3	92.3	92.3	92.3	92.3	92.3
P4	9.3	9.3	9.3	11.8	13.2	13.7	13.9	21.6	39.2	39.2	46.3	48.1	10.0	94.4	90.5	90.5

Figure 7: Scoring Mechanism in Action for r2l.

	normal	probe	dos	u2r	r2l	Acc
normal	97203	22	6	7	40	99.9%
probe	146	3961	0	0	0	96.5%
dos	20	7	3441	1	1	99.2%
u2r	13	2	0	27	10	51.9%
r2l	76	4	0	6	1040	92.4%
FP-rate	0.26%	0.88%	0.17%	34.2%	4.59%	

Figure 8: The performance of the models on $T1_{10\%}$ data-set. Note that this data-set contains very few smurf or neptune records. See text for detailed description of how this data-set is formed.

PNrule

	normal	probe	dos	u2r	r2l	Acc
normal	60316	175	75	13	14	99.5%
probe	889	3042	26	3	206	73.2%
dos	6815	57	222874	106	1	96.9%
u2r	195	3	0	15	15	6.6%
r2l	14440	12	1	6	1730	10.7%
FP-rate	27.0%	7.5%	.05%	89.5%	12.0%	
Misclassification Cost = 74058, Accuracy = 92.59%						

(a)

Contest Winner

	normal	probe	dos	u2r	r2l	Acc
normal	60262	243	78	4	6	99.5%
probe	511	3471	184	0	0	83.3%
dos	5299	1328	223226	0	0	97.1%
u2r	168	20	0	30	10	13.2
r2l	14527	294	0	8	1360	8.4%
FP-rate	25.4%	35.2%	0.1%	28.6%	1.2%	
Misclassification Cost = 72500, Accuracy = 92.71%						

(b)

Contest Runner-up

	normal	probe	dos	u2r	r2l	Acc
normal	60244	239	85	9	16	99.4%
probe	458	3521	187	0	0	84.5%
dos	5595	227	224029	2	0	97.5%
u2r	177	18	4	27	2	11.8%
r2l	14994	4	0	6	1185	7.3%
FP-rate	29.3%	21.6%	73.1%	36.4%	1.7%	
Misclassification Cost = 73243, Accuracy = 92.92%						

(c)

Results with entire test-data

PNrule

	normal	probe	dos	u2r	r2l	Acc
normal	60316	175	75	13	14	99.5%
probe	25	2349	3	0	0	98.8%
dos	392	24	222874	7	1	99.8%
u2r	22	1	0	9	7	23.1%
r2l	4248	12	1	2	1730	28.9%
FP-rate	7.2%	8.3%	.04%	71.0%	1.3%	
Misclassification Cost = 18338, Accuracy = 98.28%						

(d)

Contest Winner

	normal	probe	dos	u2r	r2l	Acc
normal	60262	243	78	4	6	99.5%
probe	0	2374	3	0	0	99.9%
dos	2	304	222992	0	0	99.9%
u2r	15	0	0	18	6	46.2%
r2l	4339	289	0	5	1360	22.7%
FP-rate	6.7%	26.0%	.03%	33.3%	0.9%	
Misclassification Cost = 18734, Accuracy = 98.19%						

(e)

Contest Runner-up

	normal	probe	dos	u2r	r2l	Acc
normal	60244	239	85	9	16	99.4%
probe	4	2370	3	0	0	99.7%
dos	10	10	223278	0	0	99.9%
u2r	19	0	0	18	2	46.2%
r2l	4804	3	0	4	1182	19.7%
FP-rate	7.4%	9.6%	.04%	41.9%	1.5%	
Misclassification Cost = 19790, Accuracy = 98.22%						

(f)

Results on subset of test-data with known subclasses

Figure 9: Comparing PNrule results with the winner and runner-up of the KDD-CUP'99 contest.

rows of perfect rules P0 and P3 are assigned adjusted scores (refer to section 2.4); decision made with P0 gets scored higher than P3, because of its larger support. For rule P1, effects of almost all N-rules are taken into account (i.e. assigned-score=accuracy-of-left-node), except for rules N6, N7, N12 and N14. Out of these N6, N7, and N14 do not satisfy support requirement, while the left node for N12 cannot be significantly distinguished from its parent ($|Z|=0.64 < \text{MinZ}$). For rule P4, most left node distributions have a low Z-number. Parent's statistics are used for such combinations. The only N-rule that has prominent effect on P4 is N12.

Figure 8 shows the accuracy and false positive rates of the models, when they are applied to the training data-set $T1_{10\%}$. These statistics are computed after combining the scores from individual classifiers in a cost-sensitive manner. As can be seen, except for u2r, all the other models have good recall (high detection rate or Accuracy) as well as good precision (low False Positive rate). The model for u2r is difficult to learn because there are only 52 training cases available for the whole class (out of close-to 5 million records). Moreover, these 52 records are further divided into four subclasses with 30, 9, 3, and 10 records each. These numbers are too small to give much statistical significance to the decisions made using them. Despite this, we are able to identify signatures for around 52% records with only 34% false positive rate. However, PNrule's performance is quite good for the r2l class, which also has very small number of records overall (1126 out of close-to 5 million). As a matter of fact, conceptually PNrule is well-suited to handle very small (but statistically significant) classes.

When these models were applied to the corrected test-data of the contest, we obtained the results shown in Figure 9(a). According to these results, our rank would be 4th among 24 contestants. This is certainly an improvement over our previous technique (figure 6(b)). We also show the results of the winner[21] and runner-up[15] entries of the contest in figures 9(b) and 9(c) respectively. As can be seen we are not very far away in misclassification cost from the winning entry. As a matter of fact, PNrule has the best detection rate for r2l among all the contestants.

The peculiar thing to observe is the large numbers in the first column of the confusion matrices. Almost all the contestants seem to have misclassified a large number of r2l and dos records as normal. This happens because there are 6% records in the test-data (18,729 out of 311,029) belonging to 17 subclasses that were *completely absent* in the training data, and none of the contestants does a good job of capturing these *unknown* subclasses.

Hence, for a fair comparison, we decided to remove these 18,729 records. For the remainder of test-data bearing *known* class labels, we show the confusion matrices of three entries in the right half of Figure 9. PNrule results are in part (d). As can be seen, PNrule performs better than other entries in terms of misclassification cost as well as accuracy. The number of records misclassified by PNrule is almost 3.7% less than the second best (part (f)). PNrule's misclassification cost penalty is about 2.2% better than the second best (part (e)).

Since we can safely assume that many contestants have applied many different techniques to solve the problem, and our PNrule method performs better than the best two, we can conclude that PNrule certainly has promise to be an effective classification technique for problems which have similar nature to the network intrusion detection problem studied in detail here.

4 Concluding Remarks and Future Research

We proposed a new framework, PNrule, for multi-class classification problem. The key novel idea used in PNrule is that of learning a rule-based model in two stages: first find P-rules to predict presence of a class and then find N-rules to predict absence of the class. We believe that this will help in overcoming the problem of small disjuncts often faced by sequential covering based algorithms. The second novel idea in PNrule is the mechanism used for scoring. It allows to selectively tune the effect of each N-rule on a given P-rule.

We have shown via a case-study in network intrusion detection, that proposed PNrule framework holds promise of performing well for classification problems, especially the ones which have a wide variation of class distributions.

The proposed framework opens up many avenues for further testing and improvement. We are currently in process of testing PNrule on various datasets from different domains. In particular, we plan to analyze its behavior for data-sets where other sequential covering problems have faced the small disjuncts problem.

One more aspect of the proposed framework needs more work, with regards to the choice of support and accuracy thresholds in each of the phases. In our case-study application, we decided these thresholds after observing the behavior of each stage for each target class. Primary factor was the size of the class being learned, and the rate at which the support and accuracy of the newly discovered rules varied. Automated techniques need to be developed that encode these heuristics.

Also, there is a possibility that despite our efforts of learning N-rules on a *collection* of P-rule covered records, the second phase can face the small disjunct problem. In such cases, possible solutions include use of pruning mechanisms or generalization the framework to a multi-phase framework.

The scoring mechanism proposed here has many possibilities of improvement. Especially when the test-data and training-data distributions are different and we already know the test-data (a scenario similar to transduction [26]), we can modify our mechanism such that the scores reflect the true probability of a class given a record, thus justifying the use of Bayes optimality rule. Another possibility to try is to use estimates of generalization errors while scoring the PN rule combinations. There is always a possibility of imposing a framework such as MetaCost[8] on top of PNrule to make it more cost-sensitive.

The possibility of replacing the scoring mechanism by a different technique can also be explored. As an example, after the P- and N-rules are learned, PNrule's classification strategy can be encoded into a functionally equivalent decision tree where P-rules and N-rules form the decisions at the nodes. This tree can then be pruned using some decision-tree pruning method (MDL [17], for example).

Currently, PNrule uses only rules with one attribute. There is nothing that restricts the framework from using rules having more conditions in conjunction, such as rules formed using frequent sets of association rules. Also, our current mechanism to form clusters of ranges for numerical attributes has scope for improvement.

References

[1] Kamal Ali and M. Pazzani. Reducing the small disjuncts problem by learning probabilistic concept descriptions. In T. Petsche, S. J. Hanson, and J. Shavlik, editors, *Computational Learning Theory and Natural Learning Systems in Knowledge Discovery and Data Mining.* MIT Press, Cambridge, Massachusettes, 1992.

[2] C. Apte, S. J. Hong, J. Lepre, S. Prasad, and B. Rosen. RAMP: Rule abstraction for modeling and prediction. Technical Report RC-20271, IBM Research Division, 1996.

[3] V. Cherkassky and F. Mulier. *Learning from Data.* John Wiley and Sons, 1998.

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

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

[6] Andrea Danyluk and Foster 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] Pedro Domingos. The RISE system: Conquering without separating. In *Proc. of Sixth IEEE International Conference on Tools with Artificial Intelligence*, pages 704–707, New Orleans, Louisiana, 1994.

[8] Pedro Domingos. MetaCost: A general method for making classifiers cost-sensitive. In *Proc. of Fifth International Conference on Knowledge Discovery and Data Mining (KDD-99)*, pages 155–164, San Diego, California, 1999.

[9] Pedro Domingos. The role of Occam's razor in knowledge discovery. *Data Mining and Knowledge Discovery*, 3(4), 1999.

[10] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[11] Charles Elkan. KDD'99 classifier learning competition. In *http://www.epsilon.com/kdd98/harvard.html*, September 1999.

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

[13] Wei Fan, Salvatore J. Stolfo, J. Zhang, and Philip K. Chan. AdaCost: Misclassification cost-sensitive boosting. In *Proc. of Sixth International Conference on Machine Learning (ICML-99)*, Bled, Slovenia, 1999. To appear.

[14] Robert 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.

[15] Itzhak Levin. Kernel miner takes second place in KDD'99 classifier learning competition. In *http://www.llsoft.com/kdd99cup.html*, October 1999.

[16] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *Proc. of Fourth International Conference on Knowledge Discovery and Data Mining (KDD-98)*, New York City, 1998.

[17] M. Mehta, J. Rissanen, and R. Agrawal. MDL-based decision tree pruning. In *Proc. of the First Int'l Conference on Knowledge Discovery and Data Mining*, pages 216–221, Montreal, Quebec, 1995.

[18] 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.

[19] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[20] S. Muggleton and C. Feng. Efficient induction of logic programs. In *Proc. of First Conference on Algorithmic Learning Theory (ALT-90)*, Ohmsha, Tokyo, 1990.

[21] Bernhard Pfahringer. Results on known classes. In *private communication with authors*, October 1999.

[22] J. Ross Quinlan. Improved estimates for the accuracy of small disjuncts. *Machine Learning*, 6(1):93–98, 1991.

[23] J. Ross Quinlan and R. M. Cameron-Jones. Oversearching and layered search in empirical learning. In *Proc. of Fourteenth International Joint Conference on Artificial Intelligence (IJCAI-95)*, pages 1019–1024, Montreal, Canada, 1995.

[24] Kapil Surlaker. Classification strategies based on association rules. In *M.S. Plan B Project Report, Department of Computer Science, University of Minnesota*, 1999.

[25] P. Turney. Cost-sensitive learning bibliography. In *http://ai.iit.nrc.ca/bibliographies/cost-sensitive.html*, 1997.

[26] Vladimir N. Vapnik. *Statistical Learning Theory*. John Wiley and Sons, 1998.

[27] Gary 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.