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

Ramesh Agarwal[*] Mahesh V. Joshi[†]

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. 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. Learning is done 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 covered 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 in their order of discovery. Using statistics of P- and N-rules on training data, we develop a mechanism to assign a score to each decision made by the classifier. Scores from all the binary classifiers are consolidated using the misclassification cost matrix to make the final decision. In this paper, we describe the details of this proposed framework. We describe how we applied our framework to a real-life network intrusion-detection dataset, supplied as part of the KDD-CUP'99 contest. Unique features of this dataset make this a challenging application. We compare the results of our approach with 23 other contestants. As an aside, we also describe how we proved that the test-data labels provided after the contest were wrong. 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. Expanded version of this paper is available at http://www.cs.umn.edu/~mjoshi/myw.html

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 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. 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[15]. 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. 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 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 goal is to discover small number of rules (low cardinality), which cover most of the positive examples of the target class (high coverage or recall) and very few of the negative examples (high accuracy or precision).

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. Here, support means the total number of records where the 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 a real-world because usually a class consists of multiple subclasses each with unique signatures. 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. 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 decreases. If the support is allowed to reduce substantially then the discovered rules 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 problem was first identified by [12]. They showed 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 [20, 6].

One remedy to avoid this problem is to use specific-to-general search techniques [7, 16], which start with each record as the most specific rule, and progressively generalize the rule-set. However, these techniques are not usually suitable for problems with large high-dimensional data-sets because their complexity scales poorly (e.g., quadratic in training set size and cubic in number of attributes [7]).

Within the framework of general-to-specific strategies, a few remedies are proposed in [12, 2] 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 [12]. This has shown to reduce error rate of small disjuncts at the cost of increased error rate of large disjuncts. Solution proposed by [2] is to assign probabilistic measures to the rules discovered in the hope of assigning lower measures to small disjuncts. Some other solutions proposed [18] 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 the 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 PNrule, a *two stage* general-to-specific framework of learning a rule-based model. 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). The key idea is to learn a set of P-rules that together cover most of the positive examples 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. This relaxation of accuracy causes some negative examples to be covered. What differentiates our method from all the previous approaches is its second stage, in which it learns N-rules that essentially remove the negative examples collectively covered by the union of all the P-rules. We believe that the existence of this stage helps in making our algorithm less sensitive to the problem of small disjuncts in the first stage. Moreover, 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. Another novel feature of our technique is its scoring mechanism. Using the statistics of P- and N-rules on training data, we develop a method to score each decision of each binary classifier. This method weighs the effect of each N-rule on each P-rule, thereby yielding PNrule a modeling flexibility. One more feature of our proposed framework is that 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. We use the 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 present a case-study in which we applied PNrule to a real-life dataset from the network intrusion detection application. Data-set was supplied as a part of KDD-CUP'99 classifier contest [11]. It contains 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. Unique features such as very wide distribution of classes and misclassification cost based evaluation, made this contest challenging. PNrule framework evolved during and after our participation in the contest. We describe how we applied it to the data-set, and compare our results with those of 23 other participants. For the subset of test-data that belonged to subclasses 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.

The rest of this paper is organized as follows. First, we review some related work. Then, we give the details of all the steps of PNrule framework in Section 2. In Section 3, we present our case study on application of PNrule to the KDD-CUP'99 intrusion detection dataset. The paper concludes with a section on some future research directions.

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[14], RAMP[3], RISE[7], RIPPER[5], and others[15].

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). RAMP[3] combines general-to-specific and specific-to-general search directions into one single step of learning in an attempt to do annealing-like optimization. It 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.

Like RAMP and RIPPER, most techniques use, as an underlying hypothesis, the 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 [8, 19]. 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. Another feature of most rule-induction techniques is that the tolerance on accuracy is quite strict. Few algorithms [7], however, 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, PNrule ensures that accuracy is regained by removing many false positives in second stage.

PNrule's model can be considered functionally equivalent to the decision-tree model. However, the primary feature of PNrule's learning strategy is that it learns N-rules on the *collection* of examples covered by *all* P-rules. Decision-tree can be thought of as learning N-rules for individual P-rules, which we believe makes it more susceptible to the small disjunct problems, because it has to learn from a smaller set of records. Moreover, PNrule's method has the ability to discover more general N-rules which span across the records covered by different P-rules.

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

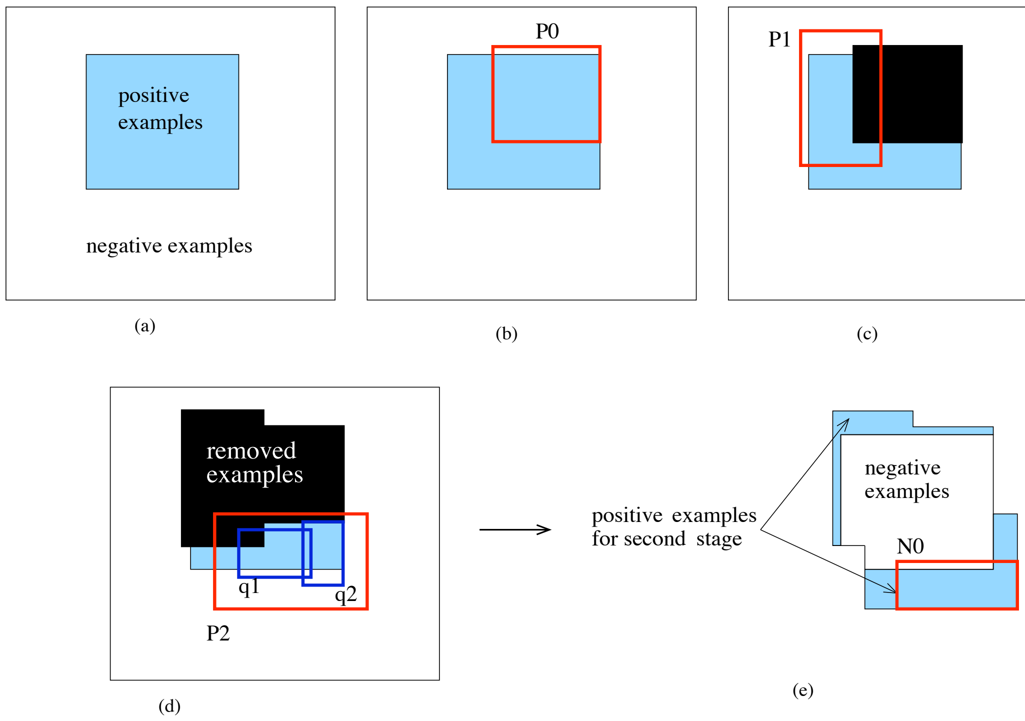


FIG. 1. *How PNrule works. (a) Original training set, (b) Discover first P-rule, (c) Discover Second P-rule on remaining examples (d) Choice of Third P-rule. P2 chosen over q1 or q2, because of its support. (e) Starting data-set for second stage.*

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 lower, then we might proceed beyond P2 to cover the remaining positive examples. Assume that we decide to stop after P2, because we have covered sufficiently large fraction of positive examples.

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 as the shaded area in Figure 1(e). Now, our goal is to learn rules that will *remove* most of these false positives. We *collect* all the examples covered by *all* the P-rules in the hope of increasing chances of finding high support and more general rules, as against learning rules to cover false positives of *individual* P-rules. So, on the dataset consisting of records covered by the union of all P-rules, we start an inverse learning process. Our new target class is now the *absence of original* target class. In Figure 1(e), the starting data-set is shown as the restricted universe and shaded area becomes 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. Iterations progress similar to the first stage. 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.* All the rules discovered during this stage are called N-rules.

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. 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

We do not describe the detailed algorithm here due to space constraints. It is given in [1]. Briefly, given a training data-set and the target class, the algorithm learns P- and N-rules using two sequential covering rule-learning stages as described in previous subsection. This is followed by a step that constructs the scoring mechanism for P-N rule combinations. The details of rule selection and the scoring mechanism are given in following subsections. Two points to note regarding the overall algorithm are as follows. First, each stage (P-stage and N-stage) of the algorithm is parametrized by support and accuracy thresholds applied to that 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. Second, if the scoring mechanism is absent, then 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 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 on a given P_i.

2.3 Choosing and Evaluating Rules

We learn very general rules. Each rule has only one condition. For a categorical attribute A, the candidate conditions are of the form $A = v$ and $A! = v$, for all possible values v of attribute A in the current training data-set S. For numerical attribute B, the conditions are of the form $B \in [low, high)$ and $B \notin [low, high)$, where the numerical range of B in S is split into multiple ranges $[low, high)$ using a simple clustering technique. Briefly, clustering starts 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 (in terms of the number of records). After this, we evaluate the strength of each range, and merge the adjacent ranges which have similar strengths. Detailed steps of the rule-selection algorithm are given in [1]. Note that the algorithm can be easily extended to discover more specific rules by adding more conditions in conjunction.

Our framework can use any performance metric that combines the distinguishing capability of a rule for the target class, support of the rule, and accuracy of the rule, all in one single metric. The metric that we used in our experiments is **Z-number**. Let a_R denote the accuracy of a given rule, R, and s_R denote its support. Let a_C denote the mean of target class C, defined as $a_C = |S_C|/|S|$, where 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 is similar to the z-test or t-test used in statistics, depending on the value of s_R. The term s_R helps in choosing high support rules. 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. Our experience with the case-study showed that this metric may have difficulty in distinguishing between highly accurate rules especially in early iterations, because it tends to give too much weight to the support. In such cases, we used another metric, which we call **Y-number**, defined as $Y_R = \sqrt{s_R}\ min(\sqrt{s_R},\ (1.0 - a_C)/(1.0 - a_R))$. The necessity of Y-number is illustrated in [1], but it should be noted that Y-number is not suitable in later iterations of the learning process, especially when a_C values are low.

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 the 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.

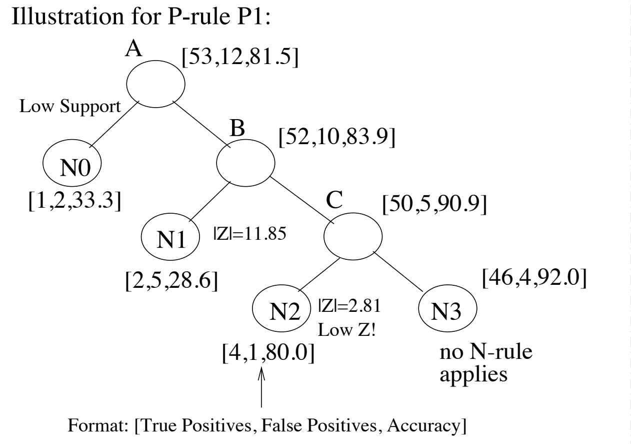
The starting point of the scoring mechanism are two matrices, SupportMatrix and ErrorMatrix. An example of these matrices is shown in Figure 2. In SupportMatrix, entry (i,j) $[j < n_N]$ gives the number of records for which the both P-rule P_i and N-rule N_j apply. Last entry in row i, SupportMatrix(i,n_N) gives the number of records where P_i applied but no N-rule applied. The ErrorMatrix records the prediction errors made by each (P_i,N_j) combination. Entries (i,j) $[j < n_N]$ give false negatives introduced by N_j for P_i's predictions, whereas (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". In figure 2, the entries in [P1,N1] location of these matrices 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 these matrices, 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.

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



Parameters:

MinSupport = 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

FIG. 2. *Illustration of Constructing the Scoring Mechanism (ScoreMatrix)*

Detailed algorithm is given in [1]. Here, we illustrate the key concepts behind it using the example given in Figure 2. 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). 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. This effect can be conceptually captured in a decision tree, as shown in the Figure for the P-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. 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 MinSupport threshold*. For our example, this 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 decided to ignore N0's effect. 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.

When N1 is applied to the new set of records (52 TP, 10 FP), it applies to 7 of those. It satisfies our support criterion of significance ($\geq$ MinSupport). 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.* 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 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. In summary, at every node of the decision tree, we apply the support and Z-number criteria to 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.

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 MinSupport, 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 $n = FP$. This is done in order to give less importance to the perfect decision made on small population as compared to the perfect decision made on larger population. Finally, the parameters MinSupport 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.

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 [9] implies that assigning x the class t which 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-to-true 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.

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. In this section, we explain the application process in detail. We start by describing the data-set and the challenges it poses.

3.1 The Data-Set and Challenges

A data-set from the network intrusion detection domain was provided as part of the KDD-CUP'99 classifier learning contest [10]. The contest problem was as follows: Given the training data-set of close to 5 million records belonging to five classes and a misclassification cost matrix, learn a classifier model so as to achieve least total misclassification cost of

		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.

predicting the labels of the supplied test-data records. The training- and test-data were collected from a controlled experiment in which a real-life military network was intentionally peppered with various attacks that hackers would use to break in. Each record in the dataset represents a connection between two network hosts. It is characterized by 41 attributes: 34 continuous-valued and 7 discrete-valued. Some examples of the attributes are duration-of-connection, number-of-bytes-transferred, number-of-failed-login-attempts, network-service-to-which-connection-was-made, etc. Each record represents 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 is quite large and it represents a real-world problem. There are four other features of the problem and data-set that made the KDD-CUP'99 contest challenging. First, the goal was not mere accuracy, but misclassification cost. The cost matrix is given in Table 1(a). Second, each attack category has some subclasses of attacks, and out of total 39 total attack subclasses that appear in test-data, only 22 were present in the training data. Third, the distribution of training records among attack categories as well as subclasses varied dramatically. Tables 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. Finally, it was told that the test-data had a completely different distribution of classes as compared to the training-data.

	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%						

FIG. 3. *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.

3.2 Our Original Strategy: Results and Critique

The PNrule framework proposed in this paper is actually an improved and automated version of a strategy we had originally developed during the three week period that was given to submit our results to the KDD-CUP'99 contest. The details of that strategy are given in [1]. Briefly, it was also a two-stage semi-automatic strategy of learning P-rules and N-rules. It had no sequential covering mechanism. Also, there was no scoring mechanism, each classifier made a pure 0-1 decision. This original strategy ranked 8th among 24 contenstants, and had the confusion matrix as shown in Figure 3(a). A very peculiar thing in the confusion matrix struck us: our false alarm rate for dos was very high. We decided to analyze it after the actual test-data labels were made available. This led us to trigger a controversy about test-data quality issue, which we briefly describe in the 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, especially given that we had found very high accuracy models 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 (among 311,029) 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% in training data-set, 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 the 2,807,886 smurf records in the training data-set. So, what are the statistical chances of a 0.002% false positive rate on this very large data-set blowing all the way upto 99.3% in the 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 a hacker's 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. We presented our arguments to the contest organizers. We were right. The correct labels in the disputed data-set (DS4) were all indeed smurf. With the new test-data, our rank improved two notches, up to 6th. The new confusion matrix is shown in Figure 3(b). As can be seen, our false alarm rate for dos is almost close to 0.0%.

3.3 Applying PNrule and Results

The sequential covering algorithms, the scoring mechanism, and cost-sensitivity make PNrule framework of this paper an improved and more automated version of our original two-stage strategy. Here is how we applied PNrule to the network intrusion detection data-set of KDD-CUP'99 contest:

1. We first developed models for smurf and neptune using the entire training set T. Then, we removed every record where smurf and neptune were predicted true with a score

greater than 99.9%. We refer to the filtered training data-set as $T1$. The filtering is done to increase the relative proportion of smaller subclasses in the training set. Moreover, the 99.9% threshold removes only those records which are strongly assured to be smurf or neptune.

2. 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 $T1_{10\%}$.

3. 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.

Detailed results for individual class models can be found in [1]. When these models were applied to the corrected test-data of the contest, we obtained the results shown in Figure 4(a). According to these results, our rank would be 4th among 24 contestants. This is certainly an improvement over our previous technique (figure 3(b)). We also show the results of the winner[17] and runner-up[13] entries of the contest in figures 4(b) and 4(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 are *completely absent* in the training data, and none of the contestants did 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 4. 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

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

FIG. 4. *Comparing PNrule results with the winner and runner-up of the KDD-CUP'99 contest.*

(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 are of similar nature as 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 the 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. Here are some aspects of the proposed framework that have scope of future research: automating selection of support and accuracy thresholds in each stage, adding some mechanisms to prevent the N-stage from running into the small disjuncts problem, improving scoring mechanism to reflect close-to-true probabilities, and improving the clustering technique used for continuous attributes.

References

[1] Ramesh Agarwal and Mahesh V. Joshi. PNrule: A new framework for learning classifier models in data mining (a case-study in network intrusion detection). Technical Report TR 00-015, Department of Computer Science, University of Minnesota, 2000.

[2] 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.

[3] 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.

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

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

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

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

[12] 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.

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

[14] 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.

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

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

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

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

[19] 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.

[20] 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.