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

Ramesh Agarwal[*] *and Mahesh V. Joshi*[†]

1 Introduction and Motivation

Learning classifier models is an important problem in data mining. Observations from the real world 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 classifier models[11]. 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.

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

[†]IBM Thomas J. Watson Research Center, P.O. Box 704, Yorktown Heights, NY 10598 (**joshim@us.ibm.com**) and Department of Computer Science, University of Minnesota, Minneapolis, MN 55455.

General form of a rule-based model is a disjunction (or union) of rules, such that 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).

Among the two broad categories of learning methods, specific-to-general techniques are not usually computationally tractable for problems with large high-dimensional data-sets that we are interested in. General-to-specific techniques are better suited for our purposes. Existing methods of this categogy usually follow a sequential covering strategy. Disjuncts in the DNF model are discovered iteratively. In each iteration, a high accuracy conjunctive rule is discovered. Then the records supported by this rule are removed, and next iteration starts with the remaining examples. These methods 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 coverage[1] of the rules decreases, which in turn causes the support[2] of the discovered rules to decrease because of the high accuracy requirements on each rule. If the support is allowed to reduce substantially then the discovered rules may be too specific to the training data, thus running into the problem of overfitting, or they may be overly general, because of various kinds of attribute and class noise present in the data [15, 5]. This leads to a greater generalization error when the model is applied to unseen cases. This problem is referred to as the problem of small disjuncts, a term first coined in [8].

Some remedies exist today to solve this problem. Some are reactive solutions. One is to stop the rule generation after the dataset size falls below some threshold. But, as noted in [8], this may miss some of the rare yet important signatures of class. Another 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 [13, 14, 4] are based on estimating a generalization capability using the training data and use it to decide whether to retain or remove a rule. One such estimate is based on minimizing the description length (MDL principle) [14, 4]. Looking at the formula for computing MDL [14], small disjuncts tend to have longer lengths because of their small support. Hence, they have a higher chance of getting removed from the final rule-set, thus possibly losing on some rare signatures.

Some proactive solutions try to avoid overfitting by pruning the rules. Existing pruning procedures in C4.5rules [14] or in RIPPER [4] work by first learning a set of most specific rules, and then generalizing each individual rule by removing some conjunctive conditions in it. In each iteration, RIPPER splits the remaining dataset into two random similar parts. One is used to grow the most specific rule and the other is used to generalize this rule immediately. As the remainder dataset size reduces, this approach may face two problems. First, the amount of training data reduces, so support of the rule decreases even further. Second, the estimates of

[1] number of positive target-class examples covered

[2] total number of examples covered, positive as well as negative

error obtained from the small pruning set may not be reliable. The strategy used by C4.5rules starts with rules obtained from an overfitted decision tree, and then uses the entire training set for generalization of each rule. Generalization is guided by pessimistic error rate estimates. However, the estimate for a small disjunct may not be reliable because of its low support. So, any decision made by comparing this estimate to the estimate of its generalized version, may be unreliable.

All these solutions can be thought of as workarounds that do not attack the root cause of the problem. We believe that the problem is caused by the low, statistically insignificant support of the small coverage rules. Having small coverage is an outcome of the presence of inherent rare subclasses and the nature of the sequential covering technique. So, it cannot be avoided. Relatively tight accuracy constraints in all the algorithms almost always cause small coverage rules to also have small support, and this causes the rules to lose their generalization ability. The key idea to overcome this problem is to allow a trade-off between support and accuracy. We believe that if accuracy constraints are relaxed gradually in later iterations, then we can keep finding rules with sufficiently large support until most of the positive examples are covered. The false positives supported during this process can be collectively removed later. Looking at it from another perspective, the problem appears because existing algorithms try to *simultaneously* optimize on recall (total coverage of the target class) as well as precision (accuracy with respect to false positives). We believe that separately conquering the two objectives of recall and precision will help in learning a model that has better generalization ability. This belief forms the foundation 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 or false positives to be supported. Now, we combine all the true positives and false positives *collectively supported by the union of* all the P-rules, and learn N-rules on this reduced data-set to remove the false positives. This two-phase approach is the first novel feature of our technique. We believe that the existence of second stage helps in making our algorithm less sensitive to the problem of small disjuncts. 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. This allows PNrule to avoid overfitting and gives it a modeling flexibility. Along with these two primary distinguishing features, PNrule is also naturally suited to take into account different costs of misclassifying different classes. 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 appli-

cation. Data-set was supplied as a part of KDDCUP'99 classifier contest [7]. Unique features such as the large size, very wide distribution of class populations, and misclassification cost based evaluation, made this contest challenging. We present two sets of results. The first set illustrates PNrule's performance on the overall multi-class classification problem as stated in the contest. We compare our results with those of 23 other participants. For the subset of test-data containing subclasses present in the training data-set, our technique performs the best in terms of accuracy as well as misclassification cost. Especially, our technique does substantially better for a class that is present in very small proportion and also carries high misclassification penalty. The second set of results illustrates PNrule's performance for binary classification of rare target classes. For this dataset, we use a more automated version of PNrule. We chose 4 rare classes from KDDCUP'99 dataset, and compared PNrule's results to those of RIPPER and C4.5rules, two state-of-the-art rule-based classifiers of its class. We show that PNrule performs significantly better on the counts of recall as well as precision, for two rare classes which benefit from the presence of second stage. For other two classes, it is as good as the best technique.

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 supported by P0, and repeat the process on the remaining set [part

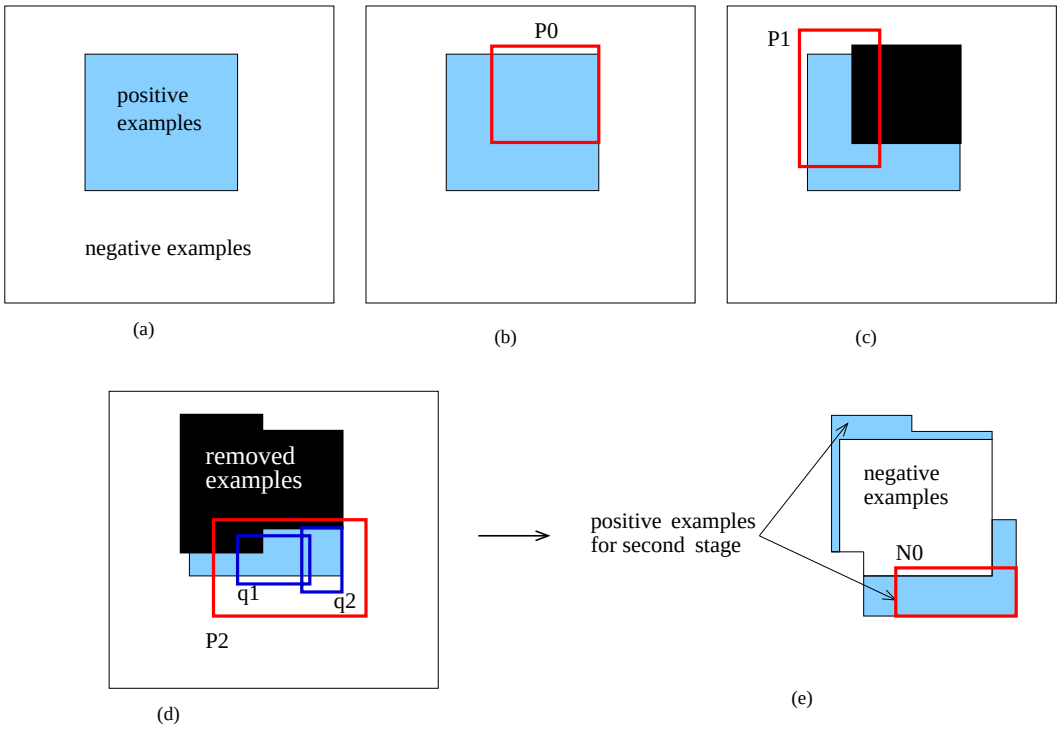


Figure 1. *How PNrule works. (a) Original training set, (b) Discover first P-rule, (c) Discover Second P-rule on remaining examples (d) Choice of Third P-rule. P2 chosen over q1 or q2, because of its support. (e) Starting data-set for second stage.*

(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 supported some 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 supported 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 supported 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 supported 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 that are still supported 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 supported 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 2.3 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 how PNrule takes different misclassification costs into account are given in [1]. The points to note here are that we use a different rule evaluation metric (called Z-number), which is based on the one-sided z-test from statistics, and we use Bayes optimality rule for combining decisions of individual binary classifiers in the best cost-effective manner. Following subsection describes a crucial part of our framework, that assigns scores to the decisions made by each binary classifier. This scoring mechanism is important in yielding PNrule a modeling flexibility. 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

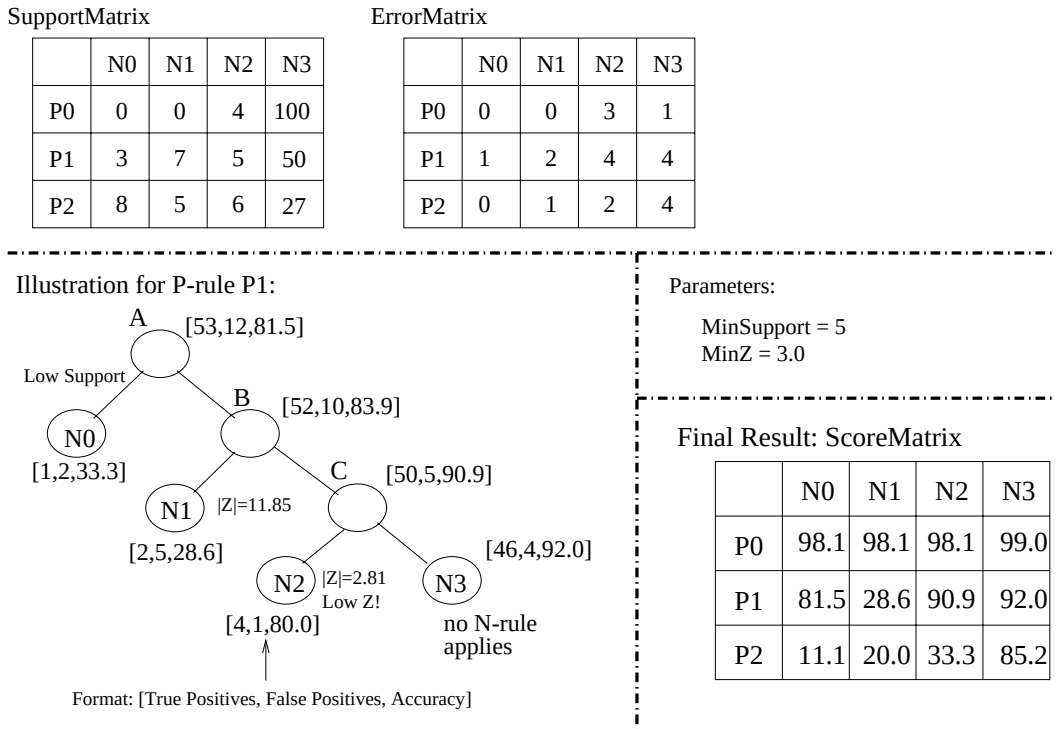
in the sense that all conjunctions have all but one conditions in common. This might seem to restrict the kinds of functions we can learn using our model.

2.3 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* supported by all P-rules. So, each N-rule is significant in removing the collective false positives. However, a given N-rule may be effective in removing false positives of only a subset of P-rules. Moreover, some low accuracy N-rule may be introducing excessive false negatives for some P-rules, possibly because its primary contribution is to remove false positives of other lower accuracy P-rules. Such excessive false negatives can be recovered by assigning them a correspondingly low score. Thus, we need to properly judge the significance of each N-rule for each P-rule.

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 supported by rule P1, rule N1 applied to 7 records (Sup-



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

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

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

portMatrix[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.

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 supported 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 similarly for N2. 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.

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. A data-set from the network intrusion detection domain was provided as part of the KDDCUP'99 classifier learning contest [6]. The contest problem was as follows: Given the training data-set of close

actual class	predicted class: normal	probe	dos	u2r	r2l
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.*

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 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 KDDCUP'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 very different distribution of classes as compared to the training-data.

3.1 Applying PNrule to the contest problem

We describe two sets of results in our case study. This section describes the application strategy and results for the multi-class classification problem as stated in the contest, which strives for a solution with least misclassification cost. Later, Section 3.2 gives results of using PNrule for binary classification of rare classes.

The PNrule framework proposed in this paper is actually an improved and

more automated version of a strategy we had originally developed during the three week period that was given to submit our results to the KDDCUP'99 contest. Based on the confusion matrix that we obtained with our original strategy and the original test data-set labels provided by the contest organizers, we triggered an interesting controversy and proved that original test-data labels were wrong. Details can be found in [1].

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 KDDCUP'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 Bayes optimality rule [1].

Note that all the rules in all the models have unit length; i.e., each P-rule and N-rule has only one condition in it. When these models were applied to the corrected test-data of the contest, we obtained the results shown in Table 2(a). We also show the results of the winner[12] and runner-up[10] entries of the contest in tables 2(b) and 2(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 [3].

Hence, for a fair comparison, we decided to remove these 18,729 records. For the remainder of test-data bearing *previously known* class labels, we show the confu-

[3]The problem of identifying such *novel* attacks in the test-data is another important line of research.

sion matrices of three entries in the right half of Table 2. 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 are of similar nature as the network intrusion detection problem studied in detail here.

3.2 Improved PNrule for Binary Classification of Rare Classes

From the results in Table 2(d), it can be seen that PNrule performs very well on one of the rare classes r2l. Its two-phase design has a potential to perform well on such classes. We decided to validate this further by comparing PNrule's binary classification performance to other existing rule-based classifiers, C4.5rules and RIPPER. We chose to build binary classifiers for four rare classes from the KDDCUP'99 dataset: probe, dos, r2l, and u2r.

First, we put in some more automations in the PNrule strategy by allowing it to learn rules with more than one conditions in conjunction. For each phase, we incorporated some stopping criteria that determine when to stop growing an individual rule, and when to stop adding more rules [9]. We also allow PNrule to use a rule-evaluation function of user's choice. Along with the Z-number described earlier, some other choices for these functions are the gain and gainratio used by C4.5 [14], information gain used by RIPPER and FOIL [3], etc. For continuous-valued attributes, we evaluate one-sided ($\leq$, $\geq$) as well as range-based decisions, and choose the one that yields best evaluation measure. There is a choice of not using range-based decisions.

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

In order to compare the performance, we used a 10% subset of the original data-set T, which we call $T_{10\%}$. This data-set is a is a representative sample of T, and was supplied as a part of the contest. Note that this data-set is different from the $T1_{10\%}$ datset defined earlier. The examples for the dos class that we use for this experiment exclude the examples of its subclasses smurf and neptune, which occur in a large percentage in $T_{10\%}$ (56.8% and 21.7%, respectively). The classes that we are modeling have following distribution: probe 0.83%, dos 0.70%, r2l 0.23%, and u2r 0.011%.

Before comparing the three techniques, we varied some parameters of each of

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

Table 2. *Comparing PNrule results with the winner and runner-up of the KDDCUP'99 contest.*

the technique, and chose the model that yielded best results on the test data. We varied following parameters of PNrule: the length of the P-rules and N-rules (1 or unlimited as decided by the stopping criterion), rule evaluation metric (Z-number, or information gain as in RIPPER), use of range-based decisions on continuous attributes. For RIPPER and C4.5, we used their default settings. However, we tested them with two training sets: as-is and stratified. In the stratified set, the total weight on the examples of the target class was made equal to the total weight on the examples of the non-target class.

Class	C4.5rules			RIPPER			PNrule		
	Rec	Prec	F	Rec	Prec	F	Rec	Prec	F
probe	99.76	99.81	0.9978	100.0	99.37	0.9968	99.61	99.56	0.9959
dos	99.94	99.94	0.9994	100.0	99.83	0.9991	100.0	99.88	0.9994
r2l	99.11	99.47	0.9929	99.47	99.56	0.9951	99.20	97.47	0.9833
u2r	42.31	61.11	0.5000	98.08	89.47	0.9358	90.38	94.00	0.9216

Table 3. *Results of binary classification on the rare classes on the $T_{10\%}$ data-set. Rec is recall in %, Prec is precision in %.*

Class	C4.5rules			RIPPER			PNrule		
	Rec	Prec	F	Rec	Prec	F	Rec	Prec	F
probe	73.04	86.38	0.7915	81.16	77.92	0.7951	89.01	82.11	**0.8542**
dos	15.50	95.55	0.2667	22.06	95.75	**0.3586**	21.74	96.68	**0.3549**
r2l	5.23	96.36	0.0993	8.33	81.85	0.1512	13.05	82.37	**0.2252**
u2r	4.82	64.71	0.0898	11.84	55.10	**0.1949**	11.40	53.06	**0.1877**

Table 4. *Results of binary classification on the rare classes on the test data-set. Rec is recall in %, Prec is precision in %. The bold-faced values indicate the best or comparably best results (highest F-measure) for each class.*

The test data used here is the test-data with all the examples. So this dataset has some examples with subclasses that are not present in $T_{10\%}$. The results of best models learnt by each technique are tabulated in Table 3 for the $T_{10\%}$ data-set, and in Table 4 for the test data-set.

Except for the class dos, PNrule yielded best results when range-based decisions were allowed for continuous attributes. All the best PNrule models used information gain criterion for evaluating the rules. The length of the P-rules was restricted to 1 for probe and r2l. It was unrestricted; i.e., the stopping criterion was trusted, for the other two classes. For all the classes, length of N-rule was set to be unrestricted.

For C4.5rules, except for u2r, the best model was obtained when unit-weight training examples were used. For RIPPER, probe and dos needed stratified training set to yield better models, whereas r2l and u2r needed unit-weight training set.

As the test-data results in Table 4 indicate, the F-measure for PNrule is substantially better for the two classes probe and r2l. For dos and u2r, they are comparable to the best of the other two techniques (RIPPER). For dos, we observed that PNrule did not need any N-rules (i.e. no second phase). So, the fact that it performs comparably to RIPPER is not surprising. For u2r, although PNrule and RIPPER are comparable, PNrule's model was slightly less complex than RIPPER's, if the total number of conditions in the rules is used as the criterion. PNrule

model consisted of 5 P-rules and 5 N-rules with 23 conjunctive conditions overall. RIPPER's model consisted 6 rules with 31 conditions overall. As a matter of fact, for probe and r2l also, PNrule yielded a less complex model (31 and 38 conditions, respectively) as compared to RIPPER (59 and 51 conditions, respectively).

Observing Table 3 indicates that the performance of all the techniques is similar on training data-set except for u2r, where only RIPPER and PNrule are similar. This fact can be used to infer that the models which do not perform well on the test data-set are running into the problem of poor generalization error because of overfitting or small disjuncts problem.

C4.5rules in general seems to be striving for good precision without giving much attention to the recall, hence its F-measure suffers substantially for all the rare classes.

These observations and results illustrate that PNrule indeed outperforms existing state-of-the-art techniques of its class, for the modeling of rare classes, at least in the situations similar to those of the case-study data-set.

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 strategy helps in overcoming the problem of small disjuncts often faced by other 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 real-world multi-classification problems with widely varying class distributions. We have also shown that especially for rare target classes, PNrule significantly outperforms other rule-based techniques. Results also illustrated that PNrule tries to avoid overfitting by yielding higher generalization capability and learning less complex models.

The proposed framework opens up many avenues for further research. We are currently in the process of analyzing the precise situations in which PNrule's two-phase approach is indeed a necessity for obtaining good classifier models. We have obtained some good insights that we will soon write about. Here are some other possibilities of future research: evaluating various stopping criteria for growing rules, automating selection of support and accuracy thresholds in each stage, adding some pruning mechanisms to further protect the N-stage from running into overfitting, improving scoring mechanism to reflect close-to-true probabilities, and finally, extending the two-phase approach to a multi-phase approach.

Bibliography

[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 RC 21719, IBM Research Report, Computer Science/Mathematics, April 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] R. M. Cameron-Jones and J. Ross Quinlan. FOIL: A midterm report. In *ECML-93, Springer-Verlag Lecture Notes in Computer Science*, number 667, Vienna, Austria, 1993.

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

[5] 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.

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

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

[8] 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.

[9] Mahesh V. Joshi, Ramesh Agarwal, and Vipin Kumar. Mining needles in a haystack: Classifying rare-classes via two-phase rule induction. Technical report, IBM Research Report, Computer Science/Mathematics, under preparation.

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

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

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

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

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

[15] 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.