An Adaptively Evolving Intrusion Detection System using Pattern Recognition Techniques

Devi Parikh, Electrical and Computer Engineering, Carnegie Mellon University

Abstract: With networking technology evolving so rapidly, computer security has been receiving a lot of attention in the recent years. Conventional intrusion detection methods in the field of computer security are anomaly detection and misuse detection – the former suffers from high false alarm rates while the latter lacks generalization capabilities and cannot detect new attack types. Pattern recognition techniques can strike a fine balance in this trade off. The goal of this work is to use pattern recognition techniques to develop a computer network intrusion detection system that adaptively evolves with changing network conditions. In order to achieve this, several sub goals were identified. Firstly, an algorithm was implemented that uses the ensemble of classifiers approach to achieve better classification rates and effective data fusion capabilities so as to combine information from multiple sources and can adapt to changing network conditions. Secondly, thorough statistical analysis of the KDD database, that is widely used in the community for demonstrating the effectiveness of pattern recognition techniques for intrusion detection, was conducted to establish several claims that render the database (as split into training and testing subsets as provided) inappropriate for pattern recognition tools. The analysis could also be used to identify potentially irrelevant features. Thirdly, techniques to tune the intrusion detection system towards minimizing the costs of the errors, and not the error rates themselves, were implemented. And lastly, a transformation was developed to use the outputs of the classifiers – multilayer perceptron (MLP) neural networks – as estimates of the posterior probabilities of an instance belonging to a particular class; which would provide better classification rates, and potentially improve the effectiveness of several classifier combination rules such as sum rule, product rule, etc. These sub goals were addressed this semester and conclusive results have been obtained that demonstrate the effectiveness of the strategies proposed to address each of these issues. Future work involves integrating these different aspects to develop a complete algorithm for adaptively evolving intrusion detection that exploits the ensemble of classifiers approach to achieve effective intrusion detection that combines information from multiple sources and is tuned towards minimizing the cost of the errors.

Paper Organization: The rest of the paper is organized as follows. The following four sections address each of the four subgoals identified above. Section 1 addresses the use of Ensemble of Classifiers Approach for Data Fusion and Adaptability, Section 2 describes the Statistical Analysis of the KDD Database, Section 3 addresses the strategy used for Cost Minimization and Section 4 addresses the transformation used for Estimating the Posterior Probabilities Based on the Outputs of the Neural Networks. Within each of these sections, the motivation behind attempting to address the issue, the goal and anticipated advantages, a discussion of counterpart approaches, the approach used, the results obtained with a comparison to counterpart approaches, along with an analysis of these results and conclusion drawn, and future direction to be pursued will be presented. Section 5 concludes and summarizes the paper.

1. Ensemble of Classifiers Approach for Data Fusion and Adaptability

1.1. Motivation:

Computer security experts combine attack evidence from different feature sets to code attack signatures [1]. Data fusion capabilities would mimic this. Network conditions are known to vary with time, as different attack types surface and as network usage patterns change. In order to maintain accurate intrusion detection under such evolving conditions, adaptability in an intrusion detection system would be favorable.

1.2. Goal and Anticipated Advantages

The goal is to introduce data fusion and adaptability capabilities in the intrusion detection system. This would ensure that complimentary information from multiple sources of information is combined to make better informed classification decisions. It is anticipated that by using an ensemble of classifiers approach, the classification performance of the system would be better than the individual sources alone, and also better than naïve concatenation of features. Also, it is anticipated that the accuracy of intrusion detection systems that do not evolve will deteriorate as network conditions change, while the ensemble of classifiers based approach that allows for adaptability will keep an intrusion detection system's accuracy consistent even with changing network conditions.

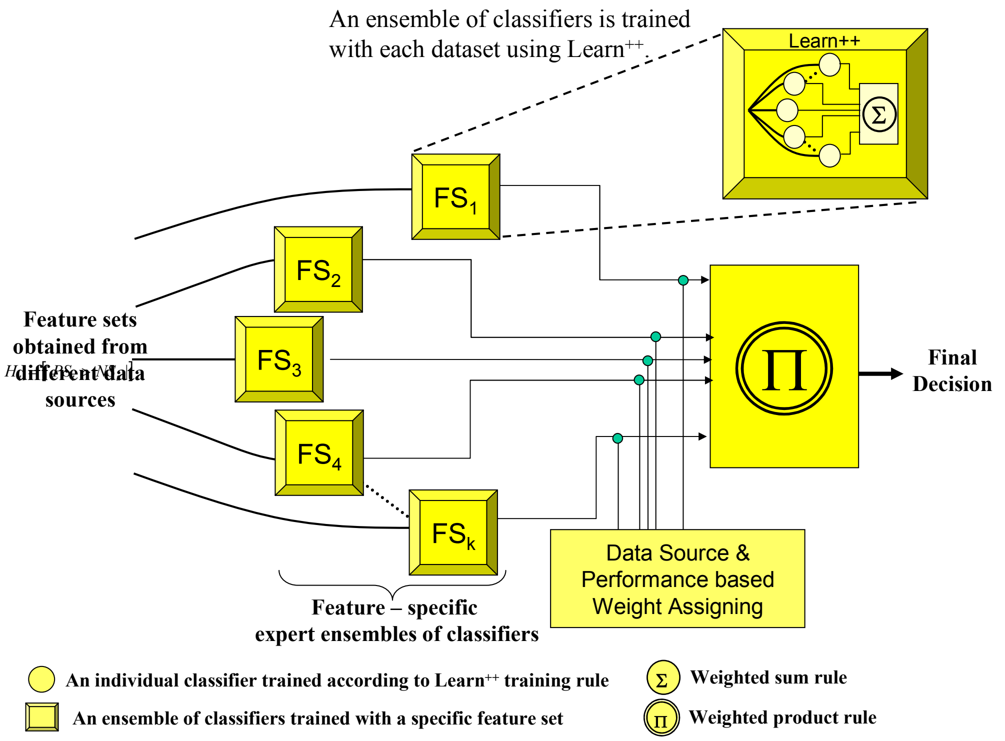
1.3. Background

The use of ensemble of classifiers for improving generalization performance has been widespread [2], however, the use the ensemble of classifiers approach to solve the problem of adaptability (similar to incremental learning) has been mostly unexplored. Various other algorithms have been suggested, such as growing or pruning of classifier architectures, selection of most informative training samples, ARTMAP algorithm, support vector machine classifiers with provable performance guarantees, kernel Hilbert spaces and adding new IF–THEN rules to an existing fuzzy inference system. These algorithms require an ad-hoc selection of a large number of parameters or require precise a priori knowledge of data distributions [3]. A similar short coming exists for the traditional data fusion algorithms based on Dempster-Schafer (DS) theory of evidence and its many variations. The majority of suggested data fusion algorithms have been developed in response to the needs of military applications, most notably target detection and tracking. More complex data fusion strategies are also widely used in practice including ensemble based variations of DS, neural networks, fuzzy systems, stacked generalization and input decimation [4]. Ensemble approaches seek to provide a fresh and a more general solution for a broader spectrum of applications. Learn++ has been shown to be capable of data fusion and incremental learning [3,5].

1.4. Approach

<u>Data fusion:</u>

An ensemble of classifiers is generated corresponding to every class. Say there are C classes, then C ensembles of classifiers are generated, each with a binary decision – positive (1): belongs to this class, negative (0): does not belong to this class. Within each ensemble, an ensemble of classifiers corresponding to each of the individual sources of information is generated. If there are K different sources of information, each of the C ensembles contain K ensembles of classifiers – with a total of $C*K$ ensembles of classifiers. MLP neural networks are used as the individual classifiers within these ensembles. There may be T_{kc} number of classifiers (MLP neural networks) in each of the $C*K$ ensembles of classifiers. These individual neural networks within the ensembles corresponding to the individual sources of information are combined using the weighted sum rule. That is, the T_{kc} classifiers within a particular ensemble among the $C*K$ ensembles are combined using the weighted sum rule thus providing a total of $C*K$ composite decisions. The decisions corresponding to the sources of information are then combined using the weighted product rule. That is, the *K* decisions within each of the *C* ensembles are combined using the weighted product rule, providing *C* final decisions. A pictorial representation of this set up (for a particular class *c*) is shown in Figure 1. When a test instance is provided, it is assigned to the class corresponding to one of the *C* ensembles of classifiers that picked the class. If multiple ensembles pick the class, the instance is assigned to the class with the highest positive score, which is explained below.



An ensemble of classifiers is trained with each dataset using Learn++.
Learn++
Σ
FS1
FS2
FS3
FS4
FSk
Feature sets obtained from different data sources
Π
Final Decision
Data Source & Performance based Weight Assigning
Feature – specific expert ensembles of classifiers
An individual classifier trained according to Learn++ training rule
An ensemble of classifiers trained with a specific feature set
Weighted sum rule
Weighted product rule

Figure 1. Pictorial representation of the DLearnin based data fusion algorithm

An algorithm inspired in part by Learn++, Dlearnin, was used to train each of the *C* ensemble of classifiers corresponding to the *C* classes. A description of Dlearnin is provided below.

As stated above, suppose there are *C* different classes. Let the different sources of information (feature sets) available be denoted as FS_k, k=1,2,…,K, where *K* is the total number of data sources. It should be noted that the feature sets available are the same for all classes. Individual classifiers for each feature set for each class are trained on a subset of the corresponding training data, randomly selected from a dynamically updated distribution over the training data instances. This distribution is biased towards those instances that have not been properly learned or seen by the previous classifiers. The block diagram, for the Dlearnin algorithm for generating the ensemble of classifiers for some class *c*, is provided in Figure 2; and described below in detail.

For each class *c*, *c* = 1,…,*C*, and for each feature set, FS_k, k=1,…,K, comprised of a different *F*eature *S*et that is submitted to Dlearnin, the inputs to the algorithm are (i) the training data S_k comprised of m_k instances $\mathbf{x}_i \in \Re^{N_k}$ along with their correct labels y_i = {+,-} or y_i = {1,0} $i=1,\ldots,m_k$ where N_k is the number of features in the k^{th} feature set; (ii) the validation data V_k

comprised of l_k instances $\mathbf{x}_i \in \Re^{N_k}$ along with their correct labels $y_i = \{+,-\}$ $i=1,...,l_k$ (iii) a supervised classification algorithm BaseClassifier, generating individual classifiers (henceforth, hypotheses); and (iv) an integer T_{kc}, the number of classifiers to be generated for the k^{th} database corresponding to the c^{th} class. The parameters below refer to the algorithm running on the k^{th} database for class c, and hence the subscripts k and c are dropped when the meaning is unambiguous to avoid subscripts of subscripts.

The BaseClassifier can be any supervised classifier such as a MLP, a support vector machine or a decision tree. MLPs were used here. A sufficiently different decision boundary can then be generated by each classifier if trained with slightly different training data subsets. Classifiers may be made relatively weak, by adjusting training parameters (such as network size and error goal) with respect to the complexity of the problem to ensure that additional diversity. However, a meaningful minimum performance is enforced: a minimum performance of 0.5 is required from the BaseClassifier i.e. it must classify atleast 50% of the instances in the validation data correctly.

During the t^{th} iteration, Dlearnin trains the BaseClassifier on a specifically selected subset of the training data to generate hypothesis h_t. The training subset TR_t is drawn from the training data according to a distribution D_t, which itself is obtained by normalizing a set of weights w_t maintained on the training data. The distribution D_t determines which instances of the training data are more likely to be selected into the training subset TR_t. Unless a priori information indicates otherwise, this distribution is initially set to be uniform, by initializing $w_1(i) = 1/m_k, \forall i = 1,\cdots,m_k$, giving equal probability to each instance to be selected into the first training subset. At each subsequent iteration t, the weights previously adjusted at iteration t-1 are normalized to ensure a legitimate probability distribution D_t (step 1):

$$\boldsymbol{D_t} = \mathbf{w_t} \Big/ \sum_{i=1}^{m_k} w_t(i) \tag{1}$$

Training subset TR_t is drawn according to D_t (step 2), and the BaseClassifier is then trained on TR_t (step 3). A hypothesis h_t is generated by the t^{th} classifier, whose error ε_t, is computed on the entire validation data V_k as the proportion of the misclassified instances (step 4)

$$\varepsilon_t = \sum_{i=1}^{l} \left[\mid h_t(x_i) \neq y_i \mid\right] \tag{2}$$

where, [| · |] evaluates to 1, if the predicate holds true, and 0 otherwise. As mentioned above, we insist that this error be less than ½. If this is the case, the hypothesis h_t is accepted and the error is normalized to obtain

$$\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}, \quad 0 < \beta_t < 1 \tag{3}$$

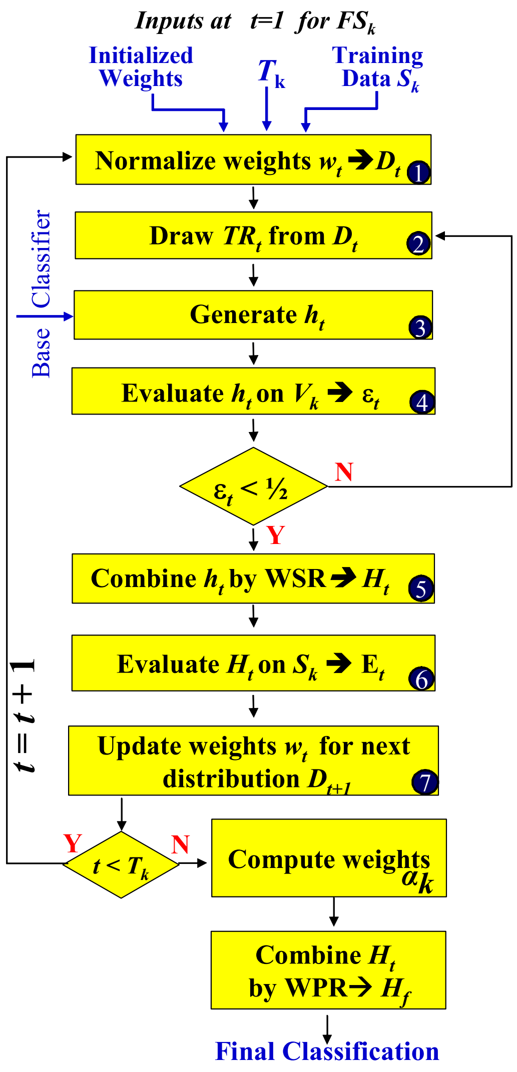


Figure 2. DLearnin block diagram

If $\varepsilon_t > ½$, then the current hypothesis is discarded, and a new training subset is selected by returning to step 2. All hypotheses generated thus far are then combined using *weighted sum rule* [6] to obtain the composite hypothesis H_t (step 5). Let the outputs of the individual classifiers generated this far be ps_t which is the positive score, that can be interpreted as the t^{th} classifier's confidence that the instance is positive (with respect to class c considering feature set k) and ns_t which is the respective negative score. ps_t and ns_t are normalized to 1. Weighted sum rule is

$$PS_t = \frac{\sum_t ps_t \log\left(\frac{1}{\beta_t}\right)}{\sum_t ps_t \log\left(\frac{1}{\beta_t}\right) + \sum_t ns_t \log\left(\frac{1}{\beta_t}\right)} \qquad NS_t = \frac{\sum_t ns_t \log\left(\frac{1}{\beta_t}\right)}{\sum_t ps_t \log\left(\frac{1}{\beta_t}\right) + \sum_t ns_t \log\left(\frac{1}{\beta_t}\right)} \qquad (4)$$

The composite hypothesis thus becomes

$$H_t = [| PS_t > NS_t |] \tag{5}$$

where H_t now represents the ensemble decision. The weighted sum rule used by Dlearnin is an undemocratic but useful combination procedure: each hypothesis is assigned a weight as the logarithm of the reciprocal of its normalized error. Therefore, those hypotheses with smaller validation error are awarded a higher voting weight and thus have more say in the final classification decision. The logarithm function is used to map a wide range of $1/\beta$ values to a smaller, more meaningful interval. The error of the composite hypothesis H_t is then computed as the sum of the distribution weights of the instances that are misclassified by the ensemble decision H_t (step 6)

$$E_t = \sum_{i:H_t(\mathbf{x}_i)\neq y_i} D_t(i) = \sum_{i=1}^{m_k} D_t(i)[| H_t(\mathbf{x}_i) \neq y_i |] \tag{6}$$

If $E_t > ½$, then the current hypothesis is discarded, and a new training subset is selected by returning to step 2. If this is not the case, the composite error is normalized to obtain

$$B_t = \frac{E_t}{1-E_t}, \quad 0 < B_t < 1 \tag{7}$$

and is used for updating the distribution weights assigned to individual instances

$$w_{t+1}(i) = w_t(i) \times B_t^{1-[|H_t(\mathbf{x}_i)\neq y_i|]} = w_t(i) \times \begin{cases} B_t\ , & if\ \ H_t(\mathbf{x}_i) = y_i \\ 1\ , & otherwise \end{cases} \tag{8}$$

Equation (8) indicates that the distribution weights of the instances correctly classified by the composite hypothesis H_t are reduced by a factor of B_t ($0<B_t<1$). Effectively, this increases the weights of the misclassified instances making them more likely to be selected to the training subset of the next iteration.

Once all the T_k classifiers are generated for feature set k, the error of the k^{th} feature sets on its validation data is computed.

$$\varepsilon_k = \sum_{i=1}^{l} [| H_T(x_i) \neq y_i |] \tag{9}$$

Since the errors of the individual classifiers on the validation data is less than ½, the composite error is also less than ½. The normalized error is calculated as

$$\alpha_k = \frac{\varepsilon_k}{1-\varepsilon_k}, 0 < \alpha_k < 1 \tag{10}$$

which will be used to determine the weight of the k^{th} feature set.

The above procedure is repeated for all K sources of information for all C classes.

When a test instance is presented, the composite hypothesis of all K feature sets of class c are combined using the *weighted product rule* to obtain the final hypothesis.

The weighted product rule is

$$PS_c = \frac{\prod_k PS_{Tk} \log\left(\frac{1}{\beta_t}\right)}{\prod_k PS_{Tk} \log\left(\frac{1}{\beta_t}\right) + \prod_k NS_{Tk} \log\left(\frac{1}{\beta_t}\right)} \qquad NS_c = \frac{\prod_k NS_{Tk} \log\left(\frac{1}{\beta_t}\right)}{\prod_k PS_{Tk} \log\left(\frac{1}{\beta_t}\right) + \prod_k NS_{Tk} \log\left(\frac{1}{\beta_t}\right)} \quad (11)$$

where PS_{Tk} and NS_{Tk} are computed using equation (4) over all classifiers generated using feature set k for class c.

The final hypothesis is then

$$H_{cfinal} = [| \, PS_c > NS_c \, |] \quad (12)$$

The test instance is assigned to the class for which H_{cfinal} is 1 (positive). If this is true for zero or more than one H_{cfinal}, the instance is assigned to the class with the highest positive score PS_c.

Conceptual similarity between data fusion and adaptability:

In many applications that call for automated decision making, it is not unusual to receive data obtained from different sources that may provide complimentary information. A suitable combination of such information is usually referred to as *data fusion*, and can lead to improved accuracy and confidence of the classification decision compared to a decision based on any of the individual data sources alone. On the other hand, it is not uncommon for testing conditions to evolve. A classification system may be exposed to data belonging to different distributions, or even different classes while it is in the field. And in order to maintain consistent classification accuracy, the system is required to evolve with the changing environment conditions. Consequently, both adaptability and data fusion involve learning from different sets of data. If the consecutive datasets that later become available arise from different classes or different distributions of data, the data fusion problem turns into an adaptability problem. Recognizing this conceptual similarity, it is anticipated that an approach similar to that used for data fusion would provide the system with adaptability capabilities.

1.5. Results and Discussion

Experiments were carried out on a subset of the database created by DARPA in the framework of the 1998 *Intrusion Detection Evaluation Program* [7]. A subset of this was pre-processed by Columbia University and distributed as part of the UCI KDD Archive and was used at the KDD 1999 cup [8] is widely used to establish the effectiveness of pattern recognition techniques for intrusion detection. The database consists of five different classes: 1 normal and 4 different attack types including probing: surveillance and other probing, e.g. port scanning, DOS: Denial of Service, R2L: unauthorized access from a remote machine e.g. guessing password, U2R: unauthorized access to local superuser (root) privileges, e.g. various "buffer overflow" attacks. However, for initial runs, the problem was treated as a two class problem: normal vs attack. Hence, $C = 2$. In the Dlearnin algorithm described above, since $C = 2$, both ensembles would be trained with the identical data and hence only one ensemble was used. The data base consists of 41 features divided into 3 feature sets: basic features, content features and traffic features. Hence, $K = 3$.

Two MLP neural networks with one hidden layer and 40 hidden layer nodes were used within each of the ensembles for each feature set for all classes. Dlearnin was run 30 times with different subsets of data used for training, validation and testing. For every run, 1000 normal instances and 1000 attack instances were randomly picked for training, 2000 each for validation and 2000 each for testing from the available ~97,000 normal instances and ~39,70,000 attack instances. For each of these splits, three different error goals were used: 0.3, 0.1 and 0.05. For each experiment, the performance of each feature set alone on the test data was noted, and the performance of data fusion using Dlearnin was noted. The averages and 95% confidence intervals of these were compared for all three different error goals. It was found that the best results were obtained (in terms of classification rate) by using the error goal of 0.05. The results corresponding to this error goal are provided in Table 1.

Table 1: Comparing classification performance of fusion with the individual feature sets

Performance	Basic	Content	Traffic	Fusion
Mean %	91.05	80.64	93.02	95.94
95% CI width	0.7175	2.5183	0.25	0.53

It can be see that the data fusion performance is better than the individual feature sets with statistical significance at 95% confidence level. This establishes the effectiveness Dlearnin to achieve efficient data fusion.

The next set of experiments were conducted using all five classes, $C = 5$. In order to compare the results using the other pattern recognition algorithms, the entire KDD database was used, split into training and testing subsets as provided. In order to make fair comparisons, the cost/instance (note: cost and cost/instance are used interchangeably) was used as the performance metric instead of the classification rate. The cost matrix provided with the KDD database is as shown below in Table 2.

Using Dlearnin, the best cost obtained was 0.2440. The cost values obtained by the entries in the KDD Cup 1999 contest are shown in Table 3 [9]. It can be seen that the value achieved by Dlearnin lies among the top 5 entries.

Table 2: Cost matrix as provided for the KDD database

Real/Predicted	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

Table 3: The cost/instance of entries in the KDD Cup 1999 contest

0.2331	0.2443	0.2531	0.2644	0.3854
0.2356	0.2474	0.2545	0.2684	0.3899
0.2367	0.2479	0.2552	0.2952	0.5053
0.2411	0.2523	0.2575	0.3344	0.9414
0.2414	0.2530	0.2588	0.3767	

Some of the other cost values obtained using MLP, a multiple classifier system made of MLP, K-means and Gaussian are 0.2424 and 0.2254 respectively [1]. Also, modular architectures were used to achieve cost values of 0.2074 and 0.1688 [1]. Two different schema for K-means were used to achieve cost values of 0.2487 and 0.2635 [10]. Several other algorithms such as Majority voting, Bayesian Average, Belief, etc. have been used that gave cost values ranging from 0.192 to 0.880 [11-14].

This shows that Dlearnin provides competitive results. However, it should be noted that in order to achieve the low cost of 0.2440, thorough classifier selection and several stages of fine tuning had to be conducted.

1.6. Future Work

The data fusion performance has been shown to be superior to the individual sources of information with statistical significance. Experiments will be run to compare the Dlearnin performance with other contemporary data fusion algorithms and naïve schemes such as concatenation of the feature sets.

An approach to be followed to introduce the adaptability capabilities has been identified. The Dlearnin algorithm will be modified to incorporate this approach. Testing scenarios to simulate changing network conditions will be simulated to test the algorithm. The performance of the system would then be tracked as the testing environment evolves and would then be compared to the performance of a conventional classification system that does not possess such adaptive capabilities. It is anticipated that the performance of the conventional classification system would deteriorate where as that of the proposed adaptive system using Dlearnin would be consistent even with changing environment conditions.

2. Statistical Analysis of the KDD Database

2.1. Motivation

While implementing Dlearnin on the KDD database, several inconsistencies were observed. In order to determine the reasons for this, statistical analysis of the database was to be conducted. It was suspected that the distributions of the training and testing data sets (as provided at the KDD Cup 1999) were different, a scenario that renders pattern recognition techniques inappropriate.

2.2. Goal and Anticipated Advantages

The goal was to use statistical tools to determine if the training and testing distributions of the KDD database were the same or not. This would enable us to pre-process the data, if required, and reduce the variability and inconsistencies in the results obtained. Also, the irrelevant features would be identified, elimination of which would reduce the dimensionality of the problem and hence facilitate learning.

2.3. Background

The analysis of variance (ANOVA) is a test that can be used to determine if the means of different populations are different with statistical significance, where the populations being compared differ from each other by some factor. ANOVA tests use the chi-squared distribution of the F parameter, where F is a measure of the ratio of inter-group variance to intra-group variance. The higher the value of F, the less likely the means are to be statistically the same. A tolerance parameter is to be specified for ANOVA testing, which specifies the probability of the means of the populations under comparison being the same. It determines how high the value of F should be to reject the null hypothesis that the means being compared are in fact the same. If this parameter is chosen to be 0.05, the conclusions drawn are statistically significant with 95% confidence.

The Fisher Ratio is defined as the ratio of inter-class scatter to intra-class scatter. Hence the higher the Fisher Ratio, the more relevant the feature is. For a two class problem the Fisher Ratio is given as

$$FR = \frac{(m_1 - m_2)^2}{v_1 + v_2} \tag{13}$$

where m_1 is the mean of class 1, m_2 is the mean of class 2, v_1 is the variance of class 1 and v_2 is the variance of class 2. For a multi-class problem, the Fisher ratio is calculated as the average of the Fisher Ratios between all possible combinations of two classes.

2.4. Approach:

The features were assumed to be independent. Based on the domain knowledge of the features, this assumption is valid. ANOVA test and the Fisher ratio were the two statistical tools that were used.

Now, the following are the two conditions that need to be satisfied in order to use pattern recognition tools effectively:

a) The distribution of the features should be statistically different for different classes
b) The distributions of the features for the training and testing data should be the same

ANOVA testing was used to identify the features that violated any of these two conditions. Central limit theorem makes the use of ANOVA testing valid. 95% confidence level was used to determine if the means of the training and testing data were statistically the same for every feature and every class. If there is less than 5% chance that the means are the same, it can be said with statistical significance that the training and testing data sets come from different statistical distributions for that feature, and thus violate condition (b). However, this can be fixed by reshuffling the data and separating them randomly into training and testing instead of using the provided split. Similarly, the means of the data coming from different classes (combined training and testing data) were compared for all feature sets using a 95% confidence level. If there is more than a 5% chance that the means are statistically the same, it can be said that the distributions of the classes are not statistically different for that feature. This violates condition (a). This indicates that the feature may be irrelevant.

The features were sorted in ascending order of their Fisher Ratio values. In order to determine which features should be eliminated, a threshold can be used. This threshold can be determined by generating an artificial feature with random values, and computing the fisher ratio for this artificial feature. All feature values with a fisher ratio less than this threshold can be identified to be irrelevant (since they are worse than random) and can thus be eliminated.

2.5. Results and Discussion

The results obtained are presented in Table 4. Column 1 shows the different features. Features 1 to 9 belong to feature set 1: basic features, features 10 to 22 belong to feature set 2: content features and features 23 to 41 belong to feature set 3: traffic features. The second column indicates the values for the Fisher Ratio. The higher the fisher ratio, the more relevant the feature is. The third column shows the rank of the features in ascending order of the fisher ratio value. Again, the higher the rank, the more relevant the feature is. The fourth column: #vb indicates how many of the five classes violate condition (b) for that feature. The fifth column #~va indicates how many of the five classes satisfy the condition (a) for that feature. The last column indicates how many of the five classes satisfy both conditions (a) and (b) for the particular feature.

Table 4: Results obtained for the statistical analysis of the KDD database

feat	Fisher Ratio	FR Rank	#vb	#~va	#~va+ #~vb
1	0.0092	14	5	3	0
2	0.3831	26	3	2	0
3	0.4809	28	3	3	1
4	0.6114	29	4	3	1
5	0.0069	12	4	0	0
6	0.0066	11	4	0	0
7	0	2	0	1	1
8	0.0022	9	1	0	0
9	0.0015	7	1	1	1
10	0.0278	17	5	0	0
11	0.0171	16	0	1	1

12	0.7799	35	4	2	1
13	0.0014	6	3	0	0
14	0.0141	15	1	3	1
15	0.0001	3	0	0	0
16	0.0006	5	2	0	0
17	0.0073	13	2	3	1
18	0.0033	10	2	1	0
19	0.0018	8	3	1	0
20	0	1	0	0	0
21	0.0005	4	1	0	0
22	0.0347	18	2	2	0
23	1.4599	41	5	3	0
24	0.9176	37	5	3	0
25	0.1019	22	5	0	0
26	0.0779	19	5	2	0
27	0.698	33	4	3	2
28	0.6213	30	4	3	2
29	0.7077	34	4	5	2
30	0.8432	36	3	1	1
31	0.1624	24	5	3	0
32	0.2321	25	3	5	3
33	1.3317	39	4	3	1
34	0.9412	38	5	5	1
35	1.3357	40	4	3	0
36	0.4152	27	4	5	1
37	0.1424	23	3	3	1
38	0.0987	21	5	0	0
39	0.0781	20	5	0	0
40	0.6745	32	5	3	2
41	0.629	31	4	3	2

It can be seen that most features either violate condition (a) or condition (b) for most of the classes. This information is summarized in Table 5. Also, it can be seen that in general, feature set 3 (features 23 to 41) have higher Fisher ratios.

Table 5: Summary of results of statistical analysis of KDD database

#~va+ #~vb	# features
0	22
1	13
2	5
3	1

It can be seen that of the 41 features, 22 features violate condition (a) or (b) for all five classes. 13 features violate one of the two conditions for 4 classes, 5 features violate either of the conditions for 3 classes and 1 feature violates one of the two conditions for 2 classes. There is no feature that satisfies both conditions for all five classes, or even four out of the five classes.

Table 6 provides more insights into the inappropriateness of the database for pattern recognition (given the particular split into training and testing).

Table 6: Further analysis of results of statistical analysis of KDD database

Metric 1	Metric 2	Corr Coef	Ideal
FR rank	#vb	0.6573	-1
FR rank	#~va	0.6744	1
Fisher Ratio	#vb	0.462	-1
Fisher Ratio	#~va	0.5584	1
#vb	#~va	0.343	-1
Fisher Ratio	# sep + con	0.3037	1
FR rank	# sep + con	0.4103	1

As stated earlier, ideally we would want features to satisfy both conditions (a) and (b). Hence, we would want features with high Fisher Ratios (or high #~va) to have low #vb. Ideally, we would want the correlation coefficient between the Fisher ratio and #vb to be -1. However, given the split of training and testing, it is seen that the higher the fisher ratio of a feature, the higher the #vb with a correlation coefficient of 0.65. Several such observations have been made in Table 6. It should be noted that Fisher Ratio and #~va are inherently similar properties of the features, and hence are highly correlated due to their basic definitions.

As stated earlier, by reshuffling the data and randomly splitting it into training and testing data subsets, the violations of condition (b) can be fixed. In order to make a quick assessment of how the classification performance improves just by reshuffling the data and using only relevant features, the following experiment was conducted.

The data was reshuffled and training and testing subsets were created randomly. Also, only feature set 3 was considered since based on the above analysis, feature set 3 was the most relevant. Classification was then performed using a simple classification technique of using the Mahanalobis distance. When a test instance is presented, the Mahanalobis distance of the instance from each of the *C* classes is computed, and the instance is assigned to the class with the least Mahanalobis distance. The Mahanalobis distance r_c of the test instance from class *c* is computed using the following equation:

$$r_c^{\,2} = (x-\mu_c)'\Sigma_c^{\,-1}(x-\mu_c) \qquad (14)$$

where *x* is the test instance, μ_c is the mean of class *c* and Σ_c is the covariance matrix of class *c*.

The cost/instance was noted. A similar test was conducted using the training and testing split as provided. It was found that the cost/instance using feature set 3, the provided split into training and testing, and Mahanalobis distance was 0.4838. When the data was reshuffled and randomly split into training and testing, under the same testing conditions, the cost/instance was found to be 0.15271 which is significantly lower. Also, it should be noted that this low cost/instance was achieved using a very simple classification technique, without any fine tuning or extensive classifier selection which is in drastic contrast with the sophisticated Dlearnin algorithm described in Section 1.4, and the intense classifier selection conducted to achieve a cost/instance of 0.2440. This shows that the given split of data into training and testing was indeed inappropriate, and much is to be gained by reshuffling the data and performing feature selection.

2.6. Future Work
The ANOVA tests will be conducted again on a random split of data to ensure that the training and testing data are representatives of each other and come from the same statistical distributions. This will also help justify the use of the statistical tools used above. Fisher Ratios need not be computed again because they are independent of the data split into training and testing. Also, the probe method described in the approach will be used to eliminate features and the performance of the classification system without these features will be compared to the performance using all the features. Also, one feature will be eliminated at a time (in ascending order of their Fisher ratios) and this will be continued till the performance of the classification system deteriorates. Physical interpretation of the features that are identified as irrelevant will be attempted to tie the statistical results to the domain knowledge.

In order to establish the effectiveness of Dlearnin in comparison with other algorithms on a different split of training and testing data than the one provided, some of the other algorithms may be implemented to simulate identical testing conditions.

Also, a smaller version of the database will be created so that tests can be simulated faster.

3. Cost Minimization

3.1. Motivation
As with several applications, the costs of the errors involved are different. A cost matrix corresponding to the different errors involved is provided with the KDD database and is shown in Table 2. All the classification strategies used so far have been geared towards maximizing the classification rate and not minimizing the cost, which is the true performance metric here.

3.2. Goal and Anticipated Advantages
The goal is to develop strategies to gear the classification system towards minimizing the cost of the errors incurred, and not the error rate itself. The anticipated advantages are that the cost incurred during final classification will be less than those incurred by systems that are not tuned towards cost minimization.

3.3. Background
For the KDD database in particular, explicit tools to minimize the cost function have not been widely explored.

3.4. Approach
In order to make classification decisions, the outputs of the classification system for a particular test instance for every possible class are noted. That is, the PS_c and NS_c values are noted for $c \in 1,\ldots,C$. If the costs of both errors (false positive and false negative) for all classes were equal, the

instance would be assigned to a particular class if $PS_c > NS_c$. If this is true for zero or multiple classes, the instance is assigned to the class with the highest PS_c value. It should be noted that the PS_c and NS_c values are normalized to 1. Hence the following two classification decisions are equivalent

```
If ($PS_c > NS_c$)
        Instance → positive
Else                                                                    (15)
        Instance → negative
```

```
If ($PS_c > 0.5$)
        Instance → positive
Else                                                                    (16)
        Instance → negative
```

Thus it can be seen that the classification decision becomes a thresholding operation on *PS*c, where the threshold is 0.5 if the costs of the two types of errors involved are equal.

However, if there are different costs associated with different errors, this may not be the optimum strategy to make classification decisions. For example for a two class problem, intuitively, if the cost of false positives is higher than false negatives, an instance should be classified as negative more frequently than as positive, in other words, the instance should be classified as positive only if the classifier is 'very confident' of its decision. If not, the instance should be classified as negative to be on the 'safe side'. Hence, when different costs are involved for the two types of errors, it is the threshold (which is 0.5 if cost of different types of errors are equal) that needs to be altered in order to minimize the cost.

In order to determine the optimum threshold to minimize the cost of errors, the cost of false positive and false negative for all the classes are to be determined. The provided cost matrix is a *C* x *C* matrix, where *C* is the number of classes. These need to be converted to *C* 2 x 2 cost matrices. In order to determine this, we need an estimate of what proportion of false negatives belonging to class *c* are assigned to class 1, class 2, … , class *C*. And then knowing the cost of these errors (as provided by the *C* x *C* cost matrix), the expected cost of a false negative associated with class *c* can be determined. This information is provided by a confusion matrix. An appropriate multiplication of the cost matrix and a normalized version of the confusion matrix (based on the validation data using the conventional threshold values of 0.5) would provide us with an estimate for the *C* 2 x 2 cost matrices. This is depicted below:

Suppose the provided *C* x *C* cost matrix is given by:

$$CstM = \begin{bmatrix} cst_{11} & cst_{12} & \dots & cst_{1C} \\ cst_{21} & cst_{22} & \dots & cst_{2C} \\ \dots & & & \\ cst_{C1} & cst_{C2} & \dots & cst_{CC} \end{bmatrix} = \begin{bmatrix} 0 & cst_{12} & \dots & cst_{1C} \\ cst_{21} & 0 & \dots & cst_{2C} \\ \dots & & & \\ cst_{C1} & cst_{C2} & \dots & 0 \end{bmatrix} \quad (17)$$

where cst_{ij} is the cost of classifying an instance that belongs to class i as class j. The cost of a correct classification is 0.

Suppose the confusion matrix on the validation data is given by

$$CnfM = \begin{bmatrix} cnf_{11} & cnf_{12} & \dots & cnf_{1C} \\ cnf_{21} & cnf_{22} & \dots & cnf_{2C} \\ \dots & & & \\ cnf_{C1} & cnf_{C2} & \dots & cnf_{CC} \end{bmatrix} \quad (18)$$

Where cnf_{ij} is the number of instances that belonged to class i but were classified as class j.

For class c, in order to determine the expected cost of false positives

$$\text{Exp}(FPcst_c) = \sum_{i=1}^{C} p_{ic} * cst_{ic} \quad (19)$$

Where p_{ic} is the probability of an instance belonging to class i being classified as class c. Since the cost of a correct classification is zero, it can be included in the summation in equation (19).

$$p_{ic} = \frac{cnf_{ic}}{\sum_{i=1}^{C} cnf_{ic}} \quad (20)$$

To reiterate, in order to determine the cost of false positives for all the C classes, the columns of the confusion matrix should be normalized to 1. Then this normalized version of the confusion matrix is multiplied (element by element) to the cost matrix. The columns of the resultant C x C matrix are summed to provide a row vector with C elements. Each of these elements provides the expected cost of false positives for the C classes. This is shown below:

Let,

$$\begin{bmatrix} 0 & fp_{12} & \dots & fp_{1C} \\ fp_{21} & 0 & \dots & fp_{2C} \\ \dots & & & \\ fp_{C1} & fp_{C2} & \dots & 0 \end{bmatrix} = \begin{bmatrix} \frac{cnf_{11}}{\sum_{i=1}^{C} cnf_{i1}} & \frac{cnf_{12}}{\sum_{i=1}^{C} cnf_{i2}} & \dots & \frac{cnf_{1C}}{\sum_{i=1}^{C} cnf_{iC}} \\ \frac{cnf_{21}}{\sum_{i=1}^{C} cnf_{i1}} & \frac{cnf_{22}}{\sum_{i=1}^{C} cnf_{i2}} & \dots & \frac{cnf_{2C}}{\sum_{i=1}^{C} cnf_{iC}} \\ \dots & & & \\ \frac{cnf_{C1}}{\sum_{i=1}^{C} cnf_{i1}} & \frac{cnf_{C2}}{\sum_{i=1}^{C} cnf_{i2}} & \dots & \frac{cnf_{CC}}{\sum_{i=1}^{C} cnf_{iC}} \end{bmatrix} .* \begin{bmatrix} 0 & cst_{12} & \dots & cst_{1C} \\ cst_{21} & 0 & \dots & cst_{2C} \\ \dots & & & \\ cst_{C1} & cst_{C2} & \dots & 0 \end{bmatrix} \quad (21)$$

where the .* operation indicates element by element multiplication. Then,

$$\begin{bmatrix} Exp(FPcst_1) & Exp(FPcst_2) & \dots & Exp(FPcst_C) \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{C} fp_{i1} & \sum_{i=1}^{C} fp_{i2} & \dots & \sum_{i=1}^{C} fp_{iC} \end{bmatrix} \quad (22)$$

A similar computation is carried out to determine the expected cost of false negatives for all the *C* classes. This is briefly summarized below:

Let

$$\begin{bmatrix} 0 & fn_{12} & \dots & fn_{1C} \\ fn_{21} & 0 & \dots & fn_{2C} \\ \dots & & & \\ fn_{C1} & fn_{C2} & \dots & 0 \end{bmatrix} = \begin{bmatrix} \frac{cnf_{11}}{\sum_{i=1}^{C} cnf_{1i}} & \frac{cnf_{12}}{\sum_{i=1}^{C} cnf_{1i}} & \dots & \frac{cnf_{1C}}{\sum_{i=1}^{C} cnf_{1i}} \\ \frac{cnf_{21}}{\sum_{i=1}^{C} cnf_{2i}} & \frac{cnf_{22}}{\sum_{i=1}^{C} cnf_{2i}} & \dots & \frac{cnf_{2C}}{\sum_{i=1}^{C} cnf_{2i}} \\ \dots & & & \\ \frac{cnf_{C1}}{\sum_{i=1}^{C} cnf_{Ci}} & \frac{cnf_{C2}}{\sum_{i=1}^{C} cnf_{Ci}} & \dots & \frac{cnf_{CC}}{\sum_{i=1}^{C} cnf_{Ci}} \end{bmatrix} .* \begin{bmatrix} 0 & cst_{12} & \dots & cst_{1C} \\ cst_{21} & 0 & \dots & cst_{2C} \\ \dots & & & \\ cst_{C1} & cst_{C2} & \dots & 0 \end{bmatrix} \quad (23)$$

then,

$$\begin{bmatrix} Exp(FNcst_1) \\ Exp(FNcst_2) \\ \dots \\ Exp(FNcst_C) \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{C} fn_{1i} \\ \sum_{i=1}^{C} fn_{2i} \\ \dots \\ \sum_{i=1}^{C} fn_{Ci} \end{bmatrix} \quad (24)$$

The resultant 2x2 cost matrix for class *c* is

$$\begin{bmatrix} 0 & Exp(FNcst_c) \\ Exp(FPcst_c) & 0 \end{bmatrix} \quad (25)$$

Having determined these *C* 2 x 2 cost matrices, the problem breaks down to finding *C* optimum thresholds for each of the *C* ensembles. The following strategy is used to determine each of the *C* thresholds.

Consider class *c*. The 2 x 2 cost matrix for class *c* provides us with the ratio of the cost of false positives over the cost of false negatives i.e. it provides us with an estimate of how much more expensive it is to classify an instance as class *c* when it is not, vs. the cost of classifying an instance that is indeed class *c*, as not class *c*. Let this ratio be r_c

$$rc = \frac{Exp(FPcst_c)}{Exp(FNcst_c)} \tag{26}$$

This ratio will determine how much the threshold should be deviated from 0.5. If for instance, the cost of false positives is greater than the cost of false negatives, i.e. $r_c > 1$, the threshold should be made higher. When the threshold is made higher, larger instances are classified as negative. This reduces the false positive rate, but it increases the false negative rate. However, if the ratio of the increase in false negative and decrease in false negative is less than r_c, increasing the threshold would still reduce the cost incurred. The threshold should be increased till the ratio of the rate of increase in false negative and rate of decrease in false positives exceeds r_c. Similarly, if $r_c < 1$, the threshold should be reduced till the ratio of increase in false positives to the decrease in false negatives exceed $1/r_c$.

In order to determine this, we need additional information other than just *r*c. We need information about the rate at which the false negatives go up/down and false positives go down/up as the threshold is increased/decreased. In order to determine this, the PS_c and NS_c values for all the instances in the validation data were noted. A histogram was plotted of the PS_c values of instances that belong to class *c* and of the instances that do not belong to class *c*. This would give us an estimate of how many false negatives increase as the threshold is increased, how many false positives increase as the threshold is decreased, and so on.

Suppose the values of the histogram of PS_c for the instances that belong to class *c* are $Nin_c(th)$ and the values of the histogram of of PS_c for the instances that donot belong to class *c* are $Nout_c(th)$. Note that the values of the $Nin_c(th)$ and $Nout_c(th)$ are for discrete values of the thresholds, at some small increments Δth.

For a given value of threshold $th = \tau$, the number of false positives is

$$\sum_{th>\tau} Nout_c(th) \tag{27}$$

And the number of false negatives is

$$\sum_{th<\tau} Nin_c(th) \tag{28}$$

Hence the cost incurred for this value of the threshold $th = \tau$ is

$$cst_c(\tau) = Exp(FPcst_c) * \sum_{th>\tau} Nout_c(th) + Exp(FNcst_c) * \sum_{th>\tau} Nin_c(th) \tag{29}$$

The optimum threshold (that minimizes the cost) is then

$$\tau_{copt\_\cos t} = \arg\min_{\tau} cst_c(\tau) \tag{30}$$

Having determined the τ_{copt_cost} for all the C classes, the classification decision is then based on these new optimum thresholds and not 0.5. The classification decisions for each of the classes then become

```
If (PSc > τcopt_cost)
        Instance → positive
Else                                                          (31a)
        Instance → negative
```

The instance is assigned to the class which satisfies the above condition. If zero or more than one class satisfy this condition (31a), the instance is assigned to the class that has the maximum $(PS_c - \tau_{copt\_cost})$ value. (31b)

To make comparisons, three different classification rules were implemented:

1) The *regular* classification rule: where the cost of errors is not considered, and a threshold of 0.5 is used to make classification decisions.

```
If (PSc > 0.5)
        Instance → positive
Else                                                          (32a)
        Instance → negative
```

If condition (32a) is satisfied for zero or more than one class, the instance is assigned to the class with the highest ($PS_c - 0.5$) value (32b)

2) The *optimum for classification* decision rule: where the cost of errors are still not considered, however the threshold for optimum classification is determined and considered while making the classification decisions. The threshold for optimum classification is determined by using equation (30) with $r_c = 1$ or $Exp(FPcst) = Exp(FNcst)$.

```
If (PSc > τcopt_classification)
        Instance → positive
Else                                                          (33a)
        Instance → negative
```

If condition (33a) is satisfied for zero or more than one class, the instance is assigned to the class with the highest ($PS_c - \tau_{copt\_classification}$) value (33b)

3) The *optimum for cost* decision rule: where the cost of errors are now considered and the threshold for minimizing the cost of errors is determined using equation (30) and considered while making the classification decisions

```
If ($PS_c > \tau_{copt\_cost}$)
        Instance → positive
Else                                                    (34a)
        Instance → negative
```

If condition (34a) is satisfied for zero or more than one class, the instance is assigned to the class with the highest ($PS_c - \tau_{copt\_cost}$) value (34b)

It is anticipated that the classification accuracy of *optimum for classification* scheme will be higher than both *regular* and *optimum for cost*, where as the cost of *optimum for cost* scheme will be lower than both *regular* and *optimum for classification*.

It should be noted that the above classification strategies are applied to the final stage after Dlearnin, when the decisions of the ensembles corresponding to the different classes H_{cfinal} according to equation (12) are to be combined to make a final decision.

3.5. Results and Discussion

The proposed strategy was first tested on artificial data. Data was generated from three different two dimensional Gaussian distributions. A total of about 16000 instances were generated – about 5333 from each class. Of these, 167 were used for training, 166 for validation and the remaining 5000 for testing. The data distribution can be seen below:

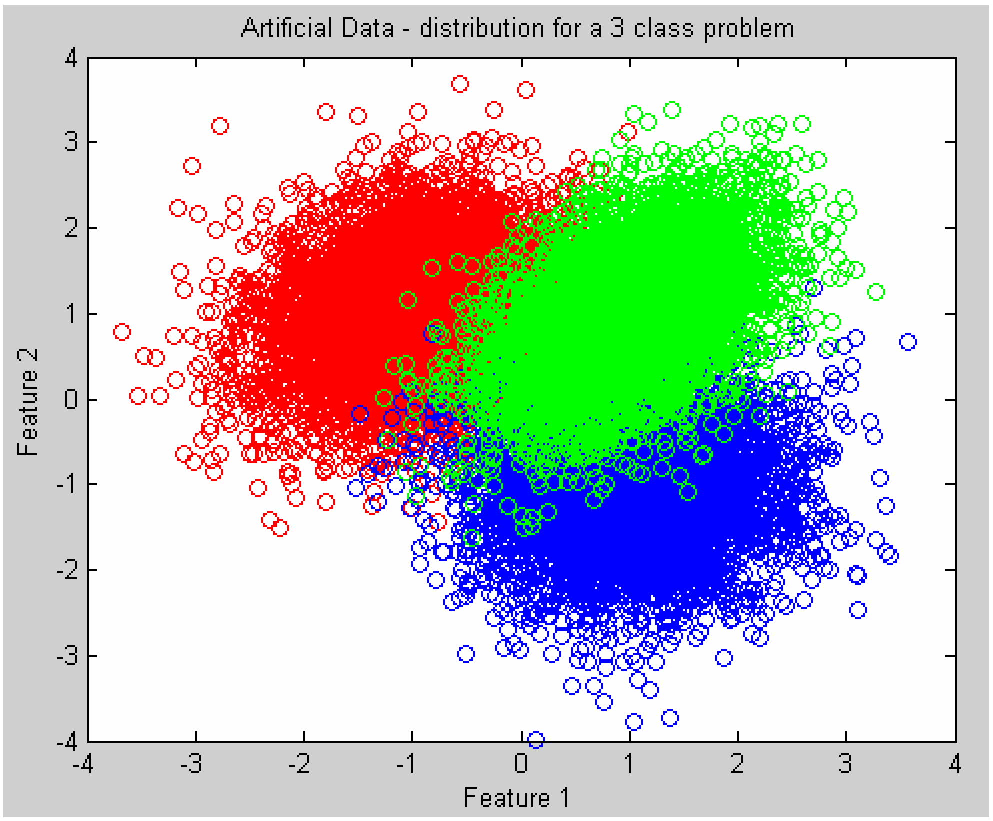


Figure 3: Artificial data distribution for a 3 class problem

The Dlearnin algorithm was implemented. Here $C = 3$ and $K = 2$. Two MLP neural networks were generated for each of the ensembles corresponding to each feature set and each class. The neural networks had one hidden layer with 2 hidden layer nodes. They were trained with an error goal of 0.07.

In order to draw statistically valid conclusions, the experiments were repeated 50 times, every time generating different data for training, validation and testing, however using the same distributions to generate the data. The results obtained are shown in Table 7.

Table 7: Comparing results obtained using three different thresholds for classification

	regular	*optimum for classification*	*optimum for cost*
Test Accuracy: Mean	90.25 %	91.68 %	89.97 %
Test Accuracy: 95% CI width	1.78 %	0.61 %	1.90 %
Test Cost: Mean	0.21	0.20	0.18
Test Cost: 95% CI width	0.026	0.020	0.016

It can be seen that as expected, the classification accuracy of *optimum for classification* is highest, while that for *optimum for cost* is the lowest. However, the cost for *optimum for cost* is lowest while that for *optimum for classification* is the highest.

3.6. Future Work

The above approach has been implemented only for an artificial database. This will now be implemented on other benchmark databases with arbitrarily chosen cost matrices and also on the KDD database where a cost matrix has been provided (Table 2).

In the above described approach, the cost matrix and confusion matrix are used to determine the optimum thresholds for minimizing cost. With these new thresholds, the confusion matrix changes, and hence a new set of thresholds can be determined using the cost matrix and the new confusion matrix. This suggests a recursive strategy to determine the optimum thresholds for minimizing the cost. This will be implemented to understand convergence issues. Also, the first iteration may be the most significant in terms of improvement, and hence analysis will be conducted to see if any significant gain is achieved in subsequent iterations.

4. Estimating the Posterior Probabilities Based on the Outputs of the Neural Networks

4.1. Motivation

In determining the optimum threshold using the strategy discussed in Section 3.4, the 2x2 cost matrices for all C classes are determined using the confusion matrix determined using the validation data. This is data dependent, and hence, a strategy that uses fewer data dependent parameters would be preferred.

If the underlying distributions of the different classes are known, Bayes classifier can be used and it would provide the optimum classification rate. However, if the cost of different errors are different, in order to minimize the expected cost, Bayes classifier would need to be modified. Suppose the cost of false positives is r times the cost of false negative. In this case, a given instance is classified as positive only if the posterior probability of the instance being positive is

r times the posterior probability of the instance being negative. In a multi-class problem, if the cost of assigning an instance belonging to class i to class j is r_{ij} times more than assigning an instance of class j to class i, then an instance is classified as class i only if

$$p_i >= r_{ij} * p_j, \text{ for all } j \in 1,\ldots,C \qquad (35)$$

Where p_n is the posterior probability that the instance belongs to class n, $n \in 1,\ldots,C$

If the outputs of the neural networks could be transformed to reflect estimates of the posterior probabilities, then a strategy similar to that explained above in equation (35) could be used to make classification decisions without explicitly determining the optimum thresholds for all the classes, and this would eliminate a portion of the data dependency.

Also, while analyzing the histograms discussed in Section 3.4, it was found that even if the costs of the different types of errors involved in the same i.e. $Exp(FPcst) = Exp(FNcst)$, the optimum threshold for classification is in fact not 0.5, and is often biased. In order to use decision rules similar to equations (15) and (16), the outputs of the neural networks should be transformed such that their optimum threshold for classification (as determined using the histograms discussed in Section 3.4) would map to 0.5.

Moreover, as discussed earlier in condition (33b), when zero or more than one class pick an instance as positive, the instance is assigned to the class with a higher $(PS_c - \tau_{copt\_classificationt})$ value. However, this may not be the optimum way. The confidence of the classifier in its decision may not be linear. This is illustrated with the following example:

Suppose $\tau_{1opt\_classification} = 0.3$, and $\tau_{2opt\_classification} = 0.8$ and for a particular test instance $PS_1 = 0.55$ and $PS_2 = 0.99$. Both classes pick the instance as positive. $(PS_1 - \tau_{1opt\_classificationt}) = 0.25$ and $(PS_2 - \tau_{2opt\_classificationt}) = 0.19$. Using the strategy described earlier, the instance would be classified as class 1. However, it is likely that an increase of 0.19 given that the optimum threshold was so high may be more significant than an increase in 0.25 for a mediocre threshold. A transform is required to understand if this non-linearity exists, and capture it if it does.

It should be noted that the strategies proposed in Section 3.4 was implemented at the final decision level, where as the transforms proposed in this Section are implemented at the individual classifier levels.

4.2. Goal and Anticipated Advantages

The goal is to develop a transform that converts the outputs of neural networks into estimates of the posterior probabilities. Although the motivation of developing such a transform to estimate the posterior probabilities based on the outputs of the neural networks lies in minimizing the cost of the classification errors, there are several anticipated advantages to this strategy

1) The classification rate would increase (because the optimum threshold determined using validation data is used for classification instead of using 0.5 which was found to be sub optimal)
2) Classifier combination rules such as the sum rule and product rule would be more effective since they require estimates of posterior probabilities

3) The cost of the classification errors will be reduced and may be more consistent and reliable because fewer data dependent parameters are involved

To reiterate, the goal is not only to increase the classification rate (this may be accomplished by several and perhaps simpler methods that consider the optimum threshold for classification). However, the goal is also to obtain better estimates of the posterior probabilities in terms of their absolute values.

4.3. Background

Two strategies are commonly used to transform the outputs of multilayer perceptron neural networks into estimates of the posterior probabilities

a) Normalization:

The outputs of the neural networks are normalized to 1, and these are often used as the estimates of the posterior probabilities. If the outputs of neural networks are *op* and *on* (these are always positive for MLP neural networks using sigmoid as the transfer function), estimates of the posterior probabilities are

$$op_{ppnorm} = \frac{op}{op + on} \quad on_{ppnorm} = \frac{on}{op + on} \tag{36}$$

b) Softmax:

If the outputs of neural networks are *op* and *on*, estimates of the posterior probabilities are

$$op_{ppsoft\max} = \frac{\exp(op)}{\exp(op) + \exp(on)} \quad on_{ppsoft\max} = \frac{\exp(on)}{\exp(op) + \exp(on)} \tag{37}$$

<u>KL Distance:</u> KL (Kullback Leibler) Distance is a measure that can be used to judge how close two probability distributions are. KL distance between two discrete distributions *p* and *q* is defined as:

$$KL(p,q) = \sum_i p(i) \log_2 \left(\frac{p(i)}{q(i)} \right) \tag{38}$$

4.4. Approach

As stated earlier, upon analysis of the histograms of *PSc* for instances that belong to class c and instances that do not belong to class c, it was found that the optimum threshold for classification (even while considering equal cost of errors) is not 0.5. Which means that *PSc* > *NSc* does not necessarily indicate that the instance has a higher probability of being positive than negative. The goal is to thus transform these PS_c values to better reflect the posterior probabilities. In order to accomplish this, the transform must satisfy the following requirements:

1) Optimum threshold should be mapped to 0.5: so that *PSc* > *NSc* indeed indicates that the instances has a higher probability of being positive than negative.
2) 0 should be mapped to 0
3) 1 should be mapped to 1

It should be noted that throughout this Section, the cost of the different types of errors are assumed to be the same and hence the optimum thresholds are the thresholds that maximize the classification rate. The only goal (so far) is to obtain better estimates of the posterior probabilities – minimizing the cost and establishing that the sum rule, product rule, etc. are more effective with the proposed transform are subsequent goals which will be addressed in future work.

A transform that satisfies these requirements, that we call the Dtransform is shown below:

$$D(o,\tau) = o^{\frac{\log(0.5)}{\log(\tau)}} \tag{39}$$

where o is the raw output of the MLP neural network, and τ is the optimum threshold for classification determined using equation (30) with $Exp(FPcst) = Exp(FNcst)$

The estimates of the posterior probabilities are then given by

$$op_{ppdtransform} = \frac{D(op,\tau_{opt})}{D(op,\tau_{opt}) + D(on,1-\tau_{opt})} \quad on_{ppdtransform} = \frac{D(on,1-\tau_{opt})}{D(op,\tau_{opt}) + D(on,1-\tau_{opt})} \tag{40}$$

In order to establish that the proposed Dtransform provides better estimates of the posterior probabilities than Normalization and Softmax, the estimates of these three methods were compared to the estimates obtained using Bayes (and normalized to 1). Bayes estimates could be determined because artificial data was used with known distributions. Three different performance measures were used to establish this fact:

1) Classification accuracy
2) Mean squared difference between the transform and Bayes estimates
3) KL distance between the distribution provided by the transform and Bayes distribution

It can be argued that the proposed Dtransform would perform better than Normalization and Softmax in all there of the above measures simply because it consider the optimum threshold values and reduces the classification error closer to the Bayes error, and hence any transform or classification decision that considers these optimum thresholds would outperform Normalization and Softmax in the above three performance metrics. In order to counter this argument, the following transform will be used to make comparisons. We call it THsoftmax – which is a modified version of Softmax, except it considers the optimum threshold as well.

$$op_{ppthsoft\max} = \frac{\exp(op-\tau_{opt})}{\exp(op-\tau_{opt}) + \exp(on-(1-\tau_{opt}))}$$

$$on_{ppthsoft\max} = \frac{\exp(on-(1-\tau_{opt}))}{\exp(op-\tau_{opt}) + \exp(on-(1-\tau_{opt}))} \tag{41}$$

Hence, comparisons were made between Normalization, Softmax, THsoftmax and Dtransform for all three performance metrics specified above.

In terms of classification accuracy, it is expected that the accuracy of Normalization and Softmax will be identical since they effectively reflect the same classification rule. However, the accuracy of THsoftmax and Dtransform should be greater than Normalization and Softmax. However, the accuracy of Dtransform may not be higher than THsoftmax with statistical significance, since they both consider the optimal threshold while making classification decisions. Inspite of this, it is expected that mean squared difference between Dtransform and Bayes estimate as well as the KL distance between Dtransform and Bayes distributions will be lower than Normalization, Softmax and even THsoftmax with statistical significance.

4.5. Results and Discussion

The above approach was implemented for a two class problem. The data was artificially generated from two dimensional Gaussian distributions and was distributed as seen below:

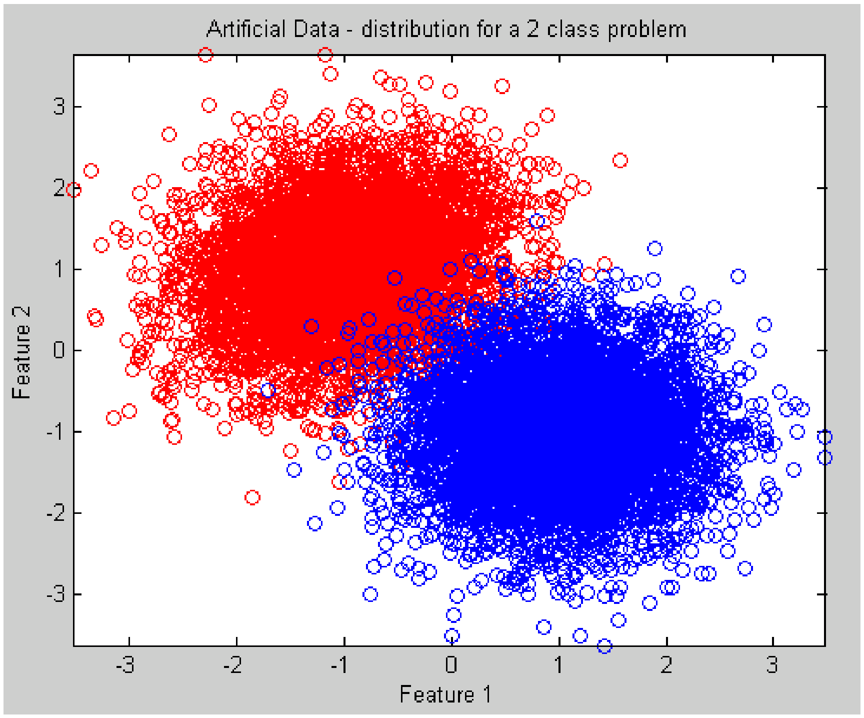


Figure 4: Artificial data distribution for a 2 class problem

Two neural networks were trained – one for each of the two classes. Each had one hidden layer, with 2 hidden layer nodes. The networks were trained with an error goal of 0.1. It was realized that while making final classification decisions, only the positive score values of the classifiers are considered. Hence, it was not evident as to why the Dtransform should map the optimal threshold (for classification, as stated earlier, the costs are assumed to be equal for the different types of errors) to 0.5, and not any other value between 0 and 1. The only requirement should be that the Dtransform should map all the optimal thresholds for different classes to a common value. This adds a parameter to the Dtransform. Let this parameter be called the *mapped_to* value. The Dtransform thus becomes:

$$D(o,\tau) = o^{\frac{\log(mapped\_to)}{\log(\tau)}} \tag{42}$$

Different mapped_to values were experimented with ranging from 0.1 to 0.9. The performance of the four transforms: Normalization, Softmax, THsoftmax and Dtransform were compared for each of the *mapped_to* values. As before, the experiments were repeated 50 times for different data sets (same distribution) and the average values are presented here. The results obtained on the test data are shown in the plots below in Figures 5, 6 and 7:

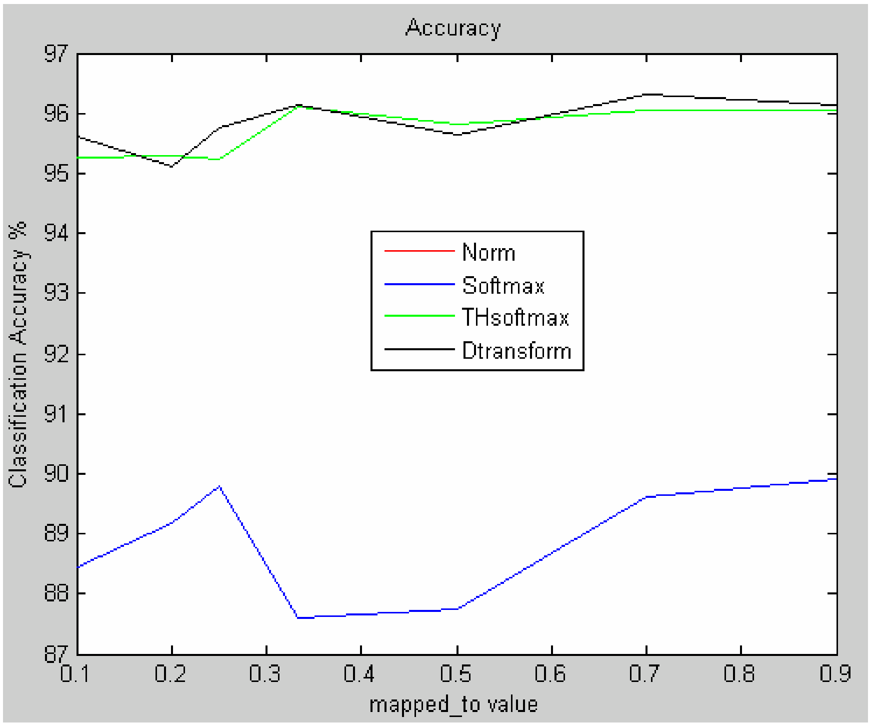


Figure 5: Classification accuracy on test data – comparing four different transforms

As expected, the accuracy for Normalization and Softmax is identical. While the accuracies for THsoftmax and Dtransform are similar. However, there is a significant increase in classification accuracy when the optimal threshold is considered – hence the accuracies of THsoftmax and Dtransform are significantly greater than those for Normalization and Softmax.

It can be seen that for *mapped_to* values less than about 0.5 the mean squared difference between the Dtransform and Bayes estimates as well as the KL distance of the Dtransform from the Bayes estimates is less than any of the other three transforms. This establishes that the Dtransform provides better estimates of the posterior probabilities than the other three transforms, for mapped_to values less than about 0.5.

4.6. Future Work

Detailed analysis is to be conducted to understand the effect of different *mapped_to* values on the performance of Dtransform. Also, similar tests have been run on three and four class problems. Also, tests have been run using just one neural network with *C* output nodes, each of which mimic the positive nodes of the *C* individual classifiers used thus far. The results obtained for these are more involved than those obtained for the two class problem, and detailed analysis is to be carried out to identify the trends in these results.

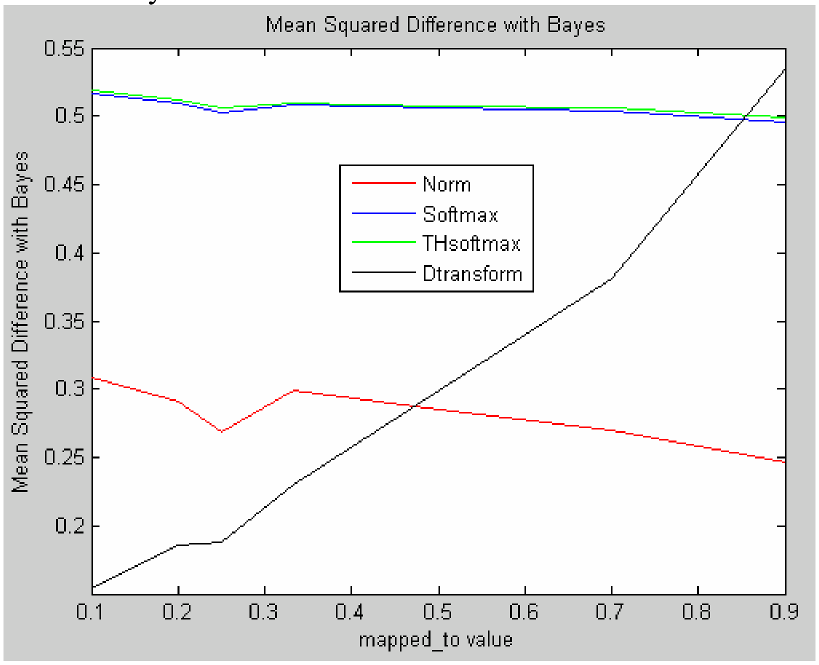


Figure 6: Mean squared difference with Bayes – comparing four different transforms

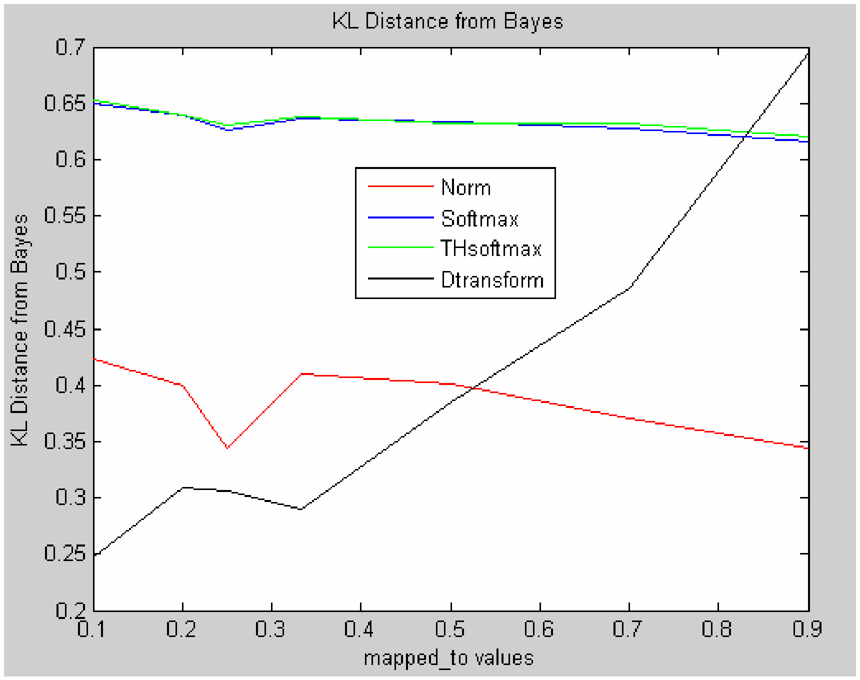


Figure 7: KL distance from Bayes – comparing four different transforms

Analysis is to be carried out to see if a range of *mapped_to* values can be identified that work well for a wide range of datasets, and theoretical explanations of why these values work well are to be investigated.

Decision rules such as those in equation (35) are to be used using the estimates provided by Dtransform to see if the cost can be significantly minimized and results obtained in Section 3.5 can be replicated using the Dtransform. Also, tests are to be run to compare the effectiveness of the sum and product rules using the estimates provided by Dtransform as opposed to those provided by Normalization, Softmax and THsoftmax.

5. Summary

To summarize, the overall goal of this work is to use ensemble of classifiers approach to achieve efficient intrusion detection what is tuned towards minimizing the cost of the errors and not the error rate itself, and that is capable of evolving with changing environment conditions. In order to achieve this, several subgoals were identified. First, efficient data fusion strategies were established in the Dlearnin algorithm which is inspired in part by Learn++. An approach has been identified that will be followed to introduce the adaptability capabilities into the classification system. Secondly, statistical analysis of the KDD database, which is widely used to establish the effectiveness of pattern recognition tools for intrusion detection, was conducted and results indicated that the split of training and testing data as provided are not good representatives of each other and come from different statistical distributions. Also several irrelevant features were identified. Eliminating the irrelevant features and reshuffling and randomly resplititng the data into training and testing subsets provided drastically better results, even with very simple classification techniques. Thirdly, strategies to gear the classification system towards minimizing the cost instead of the error rate were implemented, and convincing results were presented on artificial data. Finally, a transform was proposed to convert the outputs of MLP neural networks into estimates of the posterior probabilities and results were presented that indicate the superiority of this transform over other conventional methods in terms of estimating the posterior probabilities for artificial data. Future work involves further investigating, testing and optimizing these different techniques, introducing adaptability, and finally - integrating the different components into Dlearnin to achieve efficient intrusion detection.

References

[1] G. Giacinto, F. Roli, and L. Didaci, “A Modular Multiple Classifier System for the Detection of Intrusions in Computer Networks”, International Workshop on Multiple Classifier Systems, 2003

[2] T.G. Dietterich, “Ensemble methods in machine learning,” *Proc. 1st Int. Workshop on Multiple Classifier Systems (MCS 2000)*, LNCS vol. 1857, pp. 1 – 15, Springer: New York, NY, 2000.

[3] R. Polikar, L. Udpa, S. Udpa, V. Honavar, “Learn++: An Incremental Learning Algorithm for Supervised Neural Networks,” IEEE Trans Systems, Man and Cybernetics, 2001

[4] L. Kuncheva and C. Whitaker, “Feature Subsets for Classifier Combination: An Enumerative Experiment,” 2nd Intl Workshop on Multiple Classifier Systems, MCS 2001

[5] D. Parikh, M. Kim, J. Oagaro, S. Mandayam, R. Polikar, “Ensemble of Classifiers Approach for NDT Data Fusion” Proc. IEEE Int. Ultrasonics, Ferroelectrics and Frequency Control 50th Anniversary Joint Conf., 2004

[6] J. Kittler, M. Hatef, R. Duin, J. Matas, “On Combining Classifiers”, *IEEE Transactions on Pattern Analysis and Machnine Intelligence* 20, pp 226-239, 1998

[7] DARPA Intrusion Detection Evaluation, Lincoln Laboratory, MIT

http://www.ll.mit.edu/IST/ideval

[8] The UCI KDD Archive, Information and Computer Science, University of California, Irvine, http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[9] C. Elkan, “Results of the KDD’99 Classifier Learning”, ACM SIGKDD Explorations 1, 63-64, 2000

[10] G. Giacinto, R. Perdisci, F. Roli, “Network Intrusion Detection by Combining One Class Classifiers”, ICIAP 2005

[11] G. Giacinto and F. Roli, “Intrusion Detection in Computer Networks by Multiple Classifier Systems”, International Conference in Pattern Recognition 2002

[12] L. Didaci, G. Giacinto and F. Roli, “Ensemble Learning for Intrusion Detection in Computer Networks”, AIIA 2002 (Italian Association on Artificial Intelligence)

[13] G. Giacinto, F. Roli, and L. Didaci, “Fusion of Multiple Classifiers for Intrusion Detection in Computer Networks” Pattern Recognition Letters 2003

[14] M. Sabhnani, “Application of Machine Learning Algorithms to KDD Intrusion Detection Dataset in Misuse Detection Context”