2014 International Joint Conference on Neural Networks (IJCNN)
July 6-11, 2014, Beijing, China

Intrusion Detection Using a Cascade of Boosted Classifiers (CBC)

Mubasher Baig
Dept. of Computer Science
Lahore University of Management Sciences
Lahore, Pakistan
Email: mirza@lums.edu.pk

El-Sayed M. El-Alfy
College of Computer Sciences
King Fahd University of
Petroleum and Minerals,
Dhahran 31261, Saudi Arabia
Email: alfy@kfupm.edu.sa

Mian M. Awais
Dept. of Computer Science
Lahore University of Management Science
Lahore, Pakistan
Email: awais@lums.edu.pk

***Abstract*—A boosting-based cascade for automatic decomposition of multiclass learning problems into several binary classification problems is presented. The proposed cascade structure uses a boosted classifier at each level and use a filtering process to reduce the problem size at each level. The method has been used for detecting malicious traffic patterns using a benchmark intrusion detection dataset. A comparison of the approach with four boosting-based multiclass learning algorithms is also provided on this dataset.**

I. Introduction

Nowadays information and communication systems have become essential parts of our daily lives. However, there has been an associated increase of security threats which can cause unpleasant consequences and harms such as disclosure of confidential data, unavailability of systems and services, and unauthorized access to restricted resources. Attacks can take various forms including port scans, probes, viruses/worms, trojans, bots, rootkits, spoofing, denial of services, and exploits [1]. Intrusion detection systems are a frontier defense line against such attacks and such systems can be divided into two broad categories: signature based or anomaly based. While the first category monitor the application and traffic behavior and compares it against predefined signatures of misuses, the second category analyzes various activities to classify them as normal or intrusive activities [2]. One of the problems of signature-based approaches is the potential of high rate of false negatives due to their incapability to detect new intrusions or any slight change in the signatures of known intrusions. In contrast, anomaly-based approaches can have high rates of false positives due to detecting slight deviations from normal behavior as intrusions. Thus, to enhance the performance, signature-based approaches require frequent update of the database of known intrusions, whereas anomaly-based approaches require learning and adaptation capability. Due to their generalization and adaption characteristics, classifiers built using learning/addapting algorithms have received notable application to intrusion detection [3]. This work includes, amongst others, artificial neural networks, principal component analysis, support vector machines, and k-nearest neighbor classifiers [4], [5], [6], [7], [8], [9], [10].

The use of multiple classifier systems for intrusion detection seems more appropriate due to the distributed nature of system components and intrusion methods. Combining multiple classifiers can also solve some of the limitations of a single classifier such as over-fitting the training data and consequently improve the performance [11].

Various techniques have been proposed for constructing a highly accurate classifier from a moderately accurate learning algorithm [12], [13], [14], [15]. AdaBoost is one of the most widely used classifier learning algorithm [16] that iteratively selects multiple instances, h_t, of a classifier by modifying a weight distribution maintained on the training examples. A linear combination of the selected classifiers is then formed to generate the final prediction using

$$H(\boldsymbol{x}) = sign \sum_{t=1}^{T} \alpha_t . h_t(\boldsymbol{x}) \tag{1}$$

In its basic form, AdaBoost is a concept/binary learner hance can not handle problems involving multiple classes. Therefore, variants of AdaBoost have also been developed for multiclass problems [16], [17], [18], [19].

Some of the multiclass approaches, such as AdaBoost-M1 [16] and Multi-Class AdaBoost [19], use a multiclass base learner to handle the multiclass learning problem. AdaBoost-M1 boosts the accuracy of a multiclass base classifier by using the following modified rule to create the final ensemble:

$$H(\boldsymbol{x}) = arg \max_{y} \left(\sum_{t=1}^{T} \alpha_t . [h_t(\boldsymbol{x}) = y] \right) \tag{2}$$

This makes the maximum weight class the predicted class for x. This modification performs well with strong base classifier but it diverges when the accuracy of base classifier becomes less than 50% [19]. The Multi-Class AdaBoost modifies the computation of α_t in AdaBoost-M1 (Equation 3) such that α_t, for a k-class problem, remains positive as long as the accuracy of the base classifier is better than random guessing.

$$\alpha_t = 0.5 \left[log(\frac{\epsilon_t}{1-\epsilon_t}) + log(k-1) \right] \tag{3}$$

Another similar approach combines the probabilistic outputs of a base learner to create a multiclass ensemble. Methods using such approach estimate a probability distribution over classes for each instance x, and use it to assign a label to the instance. M-Boost [15] belongs to this class of ensemble

978-1-4799-1484-5/14/$31.00 ©2014 IEEE

creating algorithms that predict the class with highest estimated probability as the label x. Unlike most other boosting algorithms, M-Boost uses a global measure of error to reassign weights to the training examples, it computes a vector valued weight for each classifier rather than computing a single real-valued weight, and uses a different criterion for selecting a base classifier.

A completely different method of handling multiple classes decomposes a multiclass problem into several, usually orthogonal, binary classification problems each of which is then independently solved by using a binary classifier. The outputs of the learned binary classifiers are then combined to form the final multiclass classifier. Some of the well studied methods that use such a problem decomposition includes the one-vs-remaining or one-vs-one strategies, the use of error correcting codes for problem decomposition proposed by [20] and the unifying approach of Schapire [18]. In the One-vs-Remaining method of decomposing a K-class learning problem into several binary learning problems, we create one binary problem for each of the K classes. For each class a binary classification problem is created by assigning +1 label to examples of that class and the label -1 is assigned to examples in all the remaining classes. A binary classifier is learned for each of the K binary learning problems. To predict a label for an instance x, each of the K classifiers are used to compute the label of x and the class having maximum confidence/weight is finally predicted as the label of x

We observed that most of the multiclass learning algorithms tend to ignore the important but sparsely represented classes in case of a skewed dataset (i.e. a dataset that has imbalanced representation of class instances). In such cases the optimization steps in the multiclass learning algorithms tend to converge to a solution that classify only the dominant classes very accurately and still attaining a high overall accuracy. The KDD-CUP 99 dataset [21] for intrusion detection in networks is an example of such a dataset. In this dataset only three dominant classes constitute more than 98% of the total data while twenty important classes (classes representing various intrusion attacks) constitute less than 2% of the total data.

This paper presents a generic cascaded classifier based on a multiclass ensemble learning algorithm that overcomes this difficulty by decomposing the problem into several binary learning problems in adaptive fashion. The proposed cascade structure behaves like a decision tree and uses the classifiers, learned at each level, to filter the training examples reaching next level. The proposed cascaded classifier uses a filtering process similar to that of Viola et al [22] and partitions the dataset at each stage possibly eliminating some of the classes. This partitioning of the dataset at each stage results into smaller training time at successive stages. The method has been evaluated on the highly skewed KDD-cup intrusion detection dataset [21] and compared to the three boosting-based multiclass learning methods including AdaBoost-M1, Multiclass AdaBoost and M-Boost. The proposed method is also compared with a boosting based multiclass classifier obtained by using one-vs-remaining decomposition strategies.

The remaining of the paper is organized as follows. Section II introduces the multiclass M-Boost algorithm and the proposed cascade for building multiclass classifiers. Section III describes a detailed experimental setup and provides comparison with three standard boosting based algorithms including AdaBoost-M1, Multiclass AdaBoost, and M-Boost. Comparison of the proposed method with a classifier obtained using the one-vs-remaining strategy is also provided. The paper is concluded with Section IV.

II. The M-Boost Based Cascade

This section begins with a brief introduction of M-Boost [15] algorithm followed by the proposed cascade for creating classifier for handling multiclass learning problems.

A. The M-Boost Algorithm

Like AdaBoost, the M-Boost algorithm, shown in Algorithm 1, maintains a weight distribution D_t over the training examples and modifies the distribution in each round so that the misclassified examples have larger weight in the succeeding round. It also maintains a probability density over classes for each example x_i and assumes that for each instance x_i the weak classifier h_t outputs a distribution $p(c_j|x_i)$ over the k possible classes. For each instance x_i, a weighted combination of the output probabilities is used to compute distribution over the classes using .

$$p(l|x) = \frac{\sum_{t=1}^{T} \alpha_t^l . h_t^l(x)}{S. \sum_{t=1}^{T} \alpha_t^l} \tag{4}$$

where h_t^l is the probability assigned to class l by the classifier h_t, α_t^l is the weight of h_t for class l and S is the normalization factor. The final ensemble is built using the combined additive probability as given in:

$$H_T(x) = arg \max_l (p(l|x)) \tag{5}$$

where $H_T(x)$ is the class with highest probability for a given instance x. The product $\alpha_t . h_t(\bar{x})$ in the last step of M-Boost is a point by point multiplication of the two vectors and the sum is a vector sum.

B. The Cascade structure

We base the construction of proposed classifier on our observation that it might be possible to partition the training dataset into two or more sets such that an accurate classifier can be constructed to discriminate instances of each partition from the instances in remaining partitions. Once we have such a classifier available then it can be used to partition the training data such that each partition contains instances belonging to a subset of total classes. We can then recursively solve multiple smaller (smaller number of classes) problems to build the final classifier. For example, in case of the 23 class intrusion detection dataset, a partitioning of the dataset into two sets one containing the instances of the **normal** class and the other set containing the instances belonging to the remaining classes exist such that a very accurate classifier can be built using only 100 iterations of M-Boost that can discriminate instances of one set from the other. This classifier can then be used to partition training data into two sets one containing instances predicted by the classifier as normal and remaining instances predicted to be not normal. Each partition so obtained contains instances belonging to a smaller set of classes and the same partitioning process can be recursively applied to each partition

Algorithm 1 : M-Boost [15]

Require: Examples $(\bar{x_1}, y_1) \ldots (\bar{x_n}, y_n)$ where
$\bar{x_i}$ is a training instance and y_i are labels and
parameter T = total base learners in the ensemble

[Initialize Weight distributions]
set $D_1(i) = \frac{1}{n}$ $i = 1 \ldots n$ set $p_l^{\bar{x_i}} = \frac{1}{k}$ $l = 1 \ldots k$ for each $\bar{x_i}$

for t =1 to T **do**
  select h_t that minimizes error of $H_t = \sum_{j=1}^{t} \alpha_j.h_j$ w.r.t. D_t.
  Compute error ϵ_t^l of classifier h_t for each classes l
  set α_t = $(\alpha_t^1, \ldots, \alpha_t^k)$ where $\alpha_t^l = \frac{1}{2} log[(k-1)\frac{1-\epsilon_t^l}{\epsilon_t^l}]$
  [Recompute weights distribution]
  set $Entropy(x_i) = \sum_{l=1}^{k} p_l^{x_i} log(p_l^{x_i})$
  set $C_t(x_i) = \sqrt{\frac{Entropy(x_i)}{p(y_i|x_i)}}$ [Confidence]
  set $D_{t+1}(i) = \frac{exp(\alpha_t^{y_i}.C_t(x_i))}{W}$ [Reassign Weights]
  W being the normalization factor
end for
[Output the final hypothesis]
$H(\bar{x}) = \sum_{t=1}^{T} \alpha_t.h_t(\bar{x})$
$H_T(x)$ is a convex combination of the selected classifiers and outputs a distribution over classes and class with maximum probability is the predicted class.

until each partition contains instances belonging to a single class.

Based on the above observations we have devised a very simple divide-and-conquer based strategy for converting a K class problem initially into l-class ($l < k$) learning problem and then using the classifier learned for the l-class problem for reducing the problem into l subproblems each having less then K classes.

Algorithm 2 gives the details of the proposed method for building the cascade. It starts by reducing the K-class problem into a new l-class learning problem using the partitioning of classes into l sets and then uses the M-Boost algorithm to solve the resulting problem with high accuracy. The resulting classifier is saved as a node in the resulting tree-structured cascade and is used to partition the training dataset into l sets each having less than K classes. The same process is repeated for each partition independently, This divide-and-conquer process is stopped if any one of the following conditions is true:
(i) There is only one class in a partition,
(ii) The number of training instances reaching a node are less than a predefined threshold, or
(iii) Most of the examples reaching a node belong to the same class.

The **Build Cascade** algorithm (Algorithm 2) needs to be be provided with a mechanism of dividing a K-class learning problem into l-class learning problem automatically. If l is less than K then there might be exponentially many partitions of K-classes into l partitions and selection of an optimal partition by definition (i.e. a partition that yields the best error rates)

Algorithm 2 :Build Cascade

Require: Examples $(\bar{x_1}, y_1) \ldots (\bar{x_n}, y_n)$ where
$\bar{x_i}$ is a training instance and $y_i \in 1, 2, \ldots, K$ are labels and
l is the number of partitions to use
if $K > 2$ and number of training examples is greater than a threshold **then**
  Create a partition P of the K classes into l sets $P_1, P_2, \ldots, P_l$
  Create a l-class learning problem by relabeling $y_i \in P_j$ as j.
  Learn a l-class classifier M_l using M-Boost.
  Partition the training data D into l parts $D_1, D_2, \ldots, D_l$ using the predictions of M_l
  Recursively repeat the above steps for each partition
else
  Label the leaf node with the discriminating class.
end if

is NP. Therefore, in our experiments, to keep the partitioning problem tractable we always divided the K-classes into two sets at each stage one containing only one class and the second set containing the remaining classes. This made it similar to one-vs-remaining strategy with an additional filtering process. In each cascade stage, the class best discriminated from the remaining classed has been chosen as belonging to set one (+1) and all remaining classes have been placed in the second (-1) set. The resulting cascade in this case becomes a binary tree structure with the classifier best discriminating one of the classes from remaining classes used for making decision and filtering at each cascade stage. A general structure of such a cascade is shown in Figure 1(a) whereas the structure of cascade used in our experiments is shown in Figure 1(b).

Algorithm 3 :Compute Label of $\bar{x}$

Require: Instance $\bar{x}$ to be labeled
Cascaded classifier C
if C has Descendants **then**
  Use the classifier at the root of C to compute the label y of the input instance $\bar{x}$
  if $y = j$ **then**
    Recursively label the instance $\bar{x}$ by moving toward the j^th descendant of C
  end if
else
  set Label of $\bar{x}$ equal to the class label of the node.
end if

C. Using the Cascade for Classification

The hierarchical structure of the cascade offers a natural classification algorithm using the tree traversal strategy. To label an instance $\bar{x}$, we use the classifier at the root of the cascade to compute the label of $\bar{x}$ and then move along a descendent of the root corresponding to the predicted class label. This process is repeated until we reach a leaf node where the actual label of the instance x is decided. The process of assigning a label to an instance $\bar{x}$ is shown in Algorithm 3

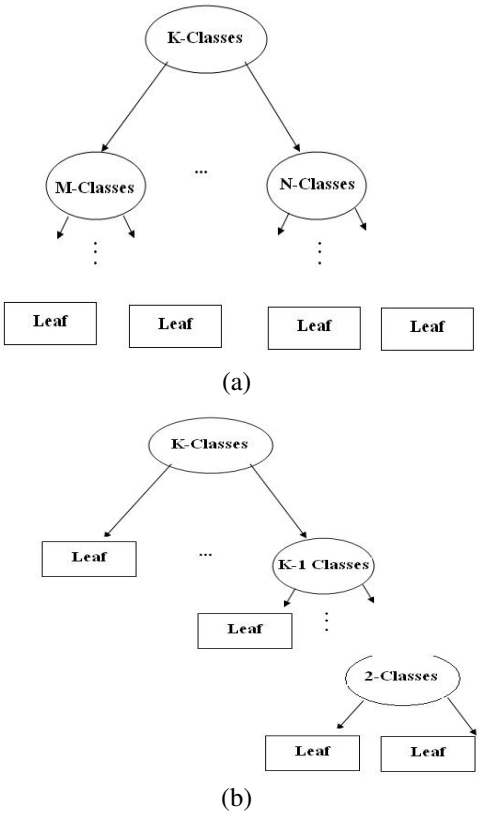

Fig. 1. Potential Cascade Structure

which uses a recursive process for assigning a label to an instance $\bar{x}$.

III. Experiments and Results

This section provides a detailed description/statistics of the dataset used in our experiments. The dataset description is followed by our experimental settings and the obtained results. The performance of the proposed multiclass cascaded structure is also compared to the results obtained using M-Boost, AdaBoost-M1, Multiclass AdaBoost, and two boosting based classifier obtained using one-vs-remaining process of multiclass to binary decomposition.

A. Dataset Description

The dataset used in our experimental work is adopted from the KDD Cup 99 (KDD'99) dataset [21] prepared and managed by MIT Lincoln Labs as part of the 1998 DARPA Intrusion Detection Evaluation Program. KDD'99 was first used for the third International Knowledge Discovery and Data Mining Tools Competition in 1999. Since then, KDD'99 has become a dominant intrusion detection dataset which has been widely used by most researchers in the machine learning community to evaluate and benchmark their work related to various types of intrusion detection [23], [24], [25], [26], [27].

The dataset consists of processed TCP dump portions of normal and attack connections to a local-area network simulating a military network environment. There are 23 different types of attack instances in the dataset falling into four

TABLE II. Class Frequency

Class	1	2	3	4	5
# of Instances	2203	30	8	53	12
Class	6	7	8	9	10
# of Instances	1247	21	9	7	107201
Class	11	12	13	14	15
# of Instances	231	97278	3	4	264
Class	16	17	18	19	20
Instance	1040	10	1589	280790	2
Class	21	22	23		
# of Instances	979	1020	20		

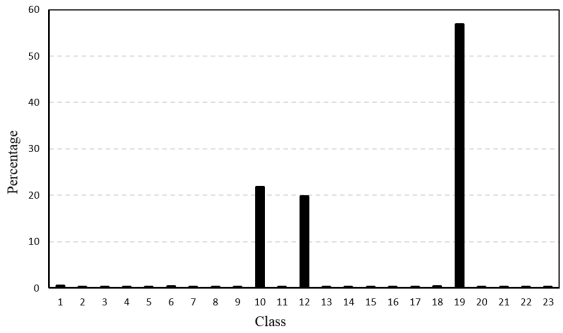

Fig. 2. Class Distribution

main categories, namely: denial of service (DoS) such as syn flooding, unauthorized access from a remote machine (R2L) such as password guessing, unauthorized access to local root privileges (U2R) such as rootkit, and probing such as port scan and nmap. The adopted dataset has 494021 connections; each described using 41 attributes and a label identifying the type of the connection (either normal or one of the attacks). Two attributes are symbolic whereas the remaining 39 attributes are binary/numeric. The attributes are divided into four groups: basic attributes of individual connections (9 attributes), content attributes within a connection suggested by domain knowledge (13 attributes), time-based traffic attributes computed using a two-second time window (9 attributes), and host-based traffic attributes computed using a window of 100 connections to the same host (10 attributes). A summary of the attributes is provided in Table I.

Detailed statistics and a percentage split of examples belonging to various classes is shown in Table-I and in Figure 2. From these statistics it is obvious that only 3 of 23 classes are dominant in the intrusion detection dataset and instances of these classes comprise more than 98% of the total training examples. This dominance of few classes pose a very interesting learning/optimization problem as most of the learning algorithms, in an effort to attain high accuracy, tend to ignore the sparse classes but still attain a very high overall accuracy.

B. Experimental Settings

In our experiments, 10-fold cross-validation has been used to estimate the performance of various multi-class learning methods over the KDD-cup dataset. The dataset was randomly split into 10 non-overlapping partitions and the training and testing are repeated 10 times using a leave one out strategy by testing the accuracy of the classifier over one of the partitions while the remaining partitions are used as the training set. It

TABLE I. SUMMARY OF VARIOUS ATTRIBUTES CATEGORY, NOTATION, NAME, TYPE, STATISTICS AND DESCRIPTION.

Cat.	Not.	Name	Type	Statistics Min	Statistics Max	Description
basic						
	a_1	duration	num.	0	58329	Connection length in seconds
	a_2	pro_type	cat.	–	–	Prototype type which can be tcp, udp, or icmp.
	a_3	srv	cat.	–	–	Service on the destination; there are 67 potential values such as http, ftp, telnet, domain, etc.
	a_4	flag	cat.	–	–	Normal or error status of the connection; there are 11 potential values, e.g. rej, sh, etc.
	a_5	src_bytes	num.	0	693M	Num. of bytes from the source to the destination
	a_6	dst_bytes	num.	0	52M	Num. of bytes from the destination to the source
	a_7	land	binary	–	–	Whether conn. from/to same host/port or not
	a_8	wrng_frg	num.	0	3	Number of wrong fragments
	a_9	urg	num.	0	3	Number of urgent packets
content						
	a_{10}	hot	num.	0	30	Number of hot indicators
	a_{11}	n_failed_lgns	num.	0	5	Number of failed login attempts
	a_{12}	logged_in	binary	–	–	Whether successfully logged in or not
	a_{13}	n_cmprmsd	num.	0	884	Number of compromised conditions
	a_{14}	rt_shell	binary	–	–	Whether root shell is obtained or not
	a_{15}	su_attmptd	num.	0	2	Number of "su root" commands attempted
	a_{16}	n_rt	num.	0	993	Number of accesses to the root
	a_{17}	n_file_crte	num.	0	28	Number of create-file operations
	a_{18}	n_shells	num.	0	2	Number of shell prompts
	a_{19}	n_access_files	num.	0	8	Number of operations on access control files
	a_{20}	n_obnd_cmds	num.	0	0	Number of outbound commands in an ftp session
	a_{21}	is_hot_lgn	binary	–	–	Whether login belongs to hot list or not
	a_{22}	is_guest_lgn	binary	–	–	Whether guest login or not
t_traffic (using a window of 2 seconds)						
	a_{23}	cnt	num.	0	511	Number of same-host connections as the current connection in the past 2 seconds
	a_{24}	srv_cnt	num.	0	511	Num. of same-host conn. to the same service as the current connection in the past 2 seconds
	a_{25}	syn_err	num.	0	1	Percentage of same-host conn. with syn errors
	a_{26}	srv_syn_err	num.	0	1	Percentage of same-service conn. with syn errors
	a_{27}	rej_err	num.	0	1	Percentage of same-host conn. with rej errors
	a_{28}	srv_rej_err	num.	0	1	Percentage of same-service conn. with rej errors
	a_{29}	sm_srv_r	num.	0	1	Percentage of same-host conn. to same service
	a_{30}	dff_srv_r	num.	0	1	Percentage of same-host conn. to different services
	a_{31}	srv_dff_hst_r	num.	0	1	Percentage of same-service conn. to different hosts
h_traffic (using a window of 100 connections)						
	a_{32}	h_cnt	num.	0	255	Number of same-host connections as the current connection in the past 100 connections
	a_{33}	h_srv_cnt	num.	0	255	Num. of same-host conn. to the same service as the current connection in the past 100 connections
	a_{34}	h_sm_srv_r	num.	0	1	Percentage of same-host conn.to same service
	a_{35}	h_dff_srv_r	num.	0	1	Percentage of same-host conn. to different services
	a_{36}	h_sm_sr_prt_r	num.	0	1	Percentage of same-service conn. to different hosts
	a_{37}	h_srv_dff_hst_r	num.	0	1	Percentage of same-service conn. to different hosts
	a_{38}	h_syn_err	num.	0	1	Percentage of same-host conn. with syn errors
	a_{39}	h_srv_syn_err	num.	0	1	Percentage of same-service conn. with syn errors
	a_{40}	h_rej_err	num.	0	1	Percentage of same-host conn. with rej errors
	a_{41}	h_srv_rej_err	num.	0	1	Percentage of same-service conn. with rej errors

is important to note that all classes had some representation in the training partition while some of the sparse classes had very little representation in the test sets. Average values of various performance measures including accuracy, precision, recall, and F-measure are reported in this paper. Decision stumps (single node decision trees) have been used as the base classifiers in all the boosting based algorithms including AdaBoost-M1, Multiclass AdaBoost and M-Boost. The reason for using decision stump learning as a base classifier is its simplicity and the ease with which the outputs of such classifiers can be converted into estimates of class probabilities. Such probability estimates are needed by the M-Boost algorithm to build the final ensemble. The conditional probability estimate, $p(c_j|x)$, of a class c_j for a given instance x was obtained by computing the ratio of the weight W_j of class j examples falling in a given partition to the total weight, W, of all examples in that partition as follows:

$$p(c_j|x) = \frac{W_j + \beta}{W + k.\beta} \tag{6}$$

The constant β in the above equation acts as a small smoothing value and is used to avoid zero probabilities and the partition boundary is determined by the decision stump.

While building the classifiers we always partitioned the undecided classes into two sets. The first set (labeled as +1) always had a single class whereas the second set consisted of all the remaining classes. For example we used the **normal** class at the root followed class labeled 19 and so on. Therefore the resulting cascade is quite similar to the classifier obtained using one-vs-remaining strategy except for the filtering step and the way the final label is assigned. The resulting cascade used in our experiments is similar to the cascade structure shown in Figure 1(b). The filtering process eliminated one of the class and the corresponding examples at each stage and

therefore fewer examples reached the succeeding stages which resulted in significantly smaller training time.

C. Results

The first set of results, shown in Table III, shows the overall training and testing performance of various learning algorithms obtained for the intrusion detection problem. The performance is measured in terms of the four commonly used performance measures including accuracy, precision, recall and F-score. The class wise values of these measures were obtained and the table presents a weighted average of these results where the weight of a class is obtained as a ratio of the number of examples of that class to the total examples in the dataset. From these results, we can see that most of the multiclass learning algorithms achieved very high values of all the four performance measures. The classifier based on the presented cascaded structure performed slightly better than the remaining classifiers.

To get a further insight into the performance of various learning algorithms, a second set of results is presented in Table IV. These results report the average values of accuracy for each class independently where the averages have been computed over the 10 results obtained using 10 folds. This set of results revels several interesting observations including the fact that although the overall accuracies are very high (98%) for the classifiers obtained using AdaBoost-M1 and M-Boost but these classifiers completely ignore most of the sparse classes and attained extremely poor performance on these important classes. It is also clear from these results that the multiclass AdaBoost, the classifier obtained using one-vs-remaining decomposition and the proposed cascaded method yielded highly accurate classifiers with the cascaded classifier having slight edge over the multiclass AdaBoost based classifier. The performance of the cascaded classifier is quite comparable to the classifier built using one-vs-remaining decomposition. The classifier built using the proposed cascade structure accurately detected the intrusion detection attacks having a very sparse representation in the dataset.

IV. Conclusions

A boosting-based cascaded classifier learning algorithm for solving multiclass learning problems has been presented. An application of the presented method for the intrusion detection dataset shows the effectiveness of proposed method for a 23 class learning problem. Comparisons of the proposed method with AdaBoost-M1, Multiclass AdaBoost, M-Boost and a boosting based classifier obtained using one-vs-remaining strategy are also provided. The performance comparison of the proposed method establishes the success of the method in building highly accurate multiclass cascaded classifiers. The method build classifiers very similar to a decision tree and is much more accurate in detecting attacks in the intrusion detection dataset than the classifiers obtained using AdaBoost-M1 and M-Boost. The performance of the method is slightly better than the classifier obtained using multiclass AdaBoost and is comparable to the performance of the classifier obtained using one-vs-remaining strategy.

Acknowledgments

The authors would like to acknowledge the support provided by the Lahore University of Management Science (LUMS), Higher Education Commission of Pakistan (HEC) and also by the King Abdulaziz City for Science and Technology (KACST) through the Science & Technology Unit at King Fahd University of Petroleum & Minerals (KFUPM) for funding this work through project No. 11-INF1658-04 as part of the National Science, Technology and Innovation Plan.

References

[1] A. Simmonds, P. Sandilands, and L. van Ekert, "An ontology for network security attacks," in *Applied Computing*. Springer, 2004, pp. 317–323.

[2] S. Axelsson, "Intrusion detection systems: A survey and taxonomy," Technical report, Tech. Rep., 2000.

[3] C.-F. Tsai, Y.-F. Hsu, C.-Y. Lin, and W.-Y. Lin, "Intrusion detection by machine learning: A review," *Expert Systems with Applications*, vol. 36, no. 10, pp. 11 994 – 12 000, 2009.

[4] S. Mukkamala, G. Janoski, and A. Sung, "Intrusion detection using neural networks and support vector machines," in *Neural Networks, 2002. IJCNN'02. Proceedings of the 2002 International Joint Conference on*, vol. 2. IEEE, 2002, pp. 1702–1707.

[5] C. Zhang, J. Jiang, and M. Kamel, "Intrusion detection using hierarchical neural networks," *Pattern Recognition Letters*, vol. 26, no. 6, pp. 779–791, 2005.

[6] J. Ryan, M.-J. Lin, and R. Miikkulainen, "Intrusion detection with neural networks," in *Advances in neural information processing systems*. MORGAN KAUFMANN PUBLISHERS, 1998, pp. 943–949.

[7] D. S. Kim and J. S. Park, "Network-based intrusion detection with support vector machines," in *Information Networking*. Springer, 2003, pp. 747–756.

[8] X. Xu and X. Wang, "An adaptive network intrusion detection method based on pca and support vector machines," in *Advanced Data Mining and Applications*. Springer, 2005, pp. 696–703.

[9] Y. Liao and V. R. Vemuri, "Use of k-nearest neighbor classifier for intrusion detection," *Computers & Security*, vol. 21, no. 5, pp. 439–448, 2002.

[10] W.-H. Chen, S.-H. Hsu, and H.-P. Shen, "Application of svm and ann for intrusion detection," *Computers & Operations Research*, vol. 32, no. 10, pp. 2617–2634, 2005.

[11] J. Kittler, M. Hatef, R. P. Duin, and J. Matas, "On combining classifiers," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 20, no. 3, pp. 226–239, 1998.

[12] R. E. Schapire, "The boosting approach to machine learning: An overview," *LECTURE NOTES IN STATISTICS-NEW YORK-SPRINGER VERLAG-*, pp. 149–172, 2003.

[13] E. Bauer and R. Kohavi, "An empirical comparison of voting classification algorithms: Bagging, boosting, and variants," *Machine learning*, vol. 36, no. 1-2, pp. 105–139, 1999.

[14] T. G. Dietterich, "Ensemble methods in machine learning," in *Multiple classifier systems*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000, vol. 1857, pp. 1–15.

[15] M. Baig and M. M. Awais, "Global reweighting and weight vector based strategy for multiclass boosting," in *International Conference on Neurual Information Processing (ICONIP 2012)*.

[16] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[17] R. E. Schapire, "The strength of weak learnability," *Machine learning*, vol. 5, no. 2, pp. 197–227, 1990.

[18] ——, "Using output codes to boost multiclass learning problems," in *Proceedings of Fourth International Conference on Machine Learning*, vol. 97, 1997, pp. 313–321.

[19] J. Zhu, H. Zou, S. Rosset, and T. Hastie, "Multi-class adaboost," *Statistics and Its Interface*, pp. 349–360, 2009.

TABLE III. TRAINING AND TEST PERFORMANCE COMPARISON OF VARIOUS METHODS

Phase	Method	Accuracy	Precision	Recall	F_1 Measure
Training					
	AdaBoost-M1	0.991 ± 0.0004	0.964 ± 0.0003	0.973 ± 0.005	0.96 ± 0.007
	Multiclass AdaBoost	0.999 ± 0.0001	0.998 ± 0.0003	0.998 ± 0.003	0.997 ± 0.008
	M-Boost	0.990 ± 0.0004	0.964 ± 0.0003	0.97 ± 0.005	0.96 ± 0.007
	Cascaded M-Boost	1 ± 0.0001	0.999 ± 0.0001	0.999 ± 0.0013	0.999 ± 0.002
	One-vs-Remaining	1	1	0.999	0.999
Testing					
	AdaBoost-M1	0.989 ± 0.001	0.957 ± 0.004	0.964 ± 0.004	0.961 ± 0.006
	Multiclass AdaBoost	0.998 ± 0.006	0.997 ± 0.004	0.997 ± 0.005	0.996 ± 0.007
	M-Boost	0.989 ± 0.001	0.957 ± 0.004	0.964 ± 0.004	0.961 ± 0.006
	Cascaded M-Boost	0.999 ± 0.0001	0.998 ± 0.0003	0.999 ± 0.003	0.998 ± 0.003
	One-vs-Remaining	0.999	0.998	0.998	0.998

TABLE IV. CLASS-WISE COMPARISON OF ACCURACY OVER TEST-DATA

Class	AdaBoost-M1	M-Boost	Multi-class AdaBoost	Cascaded AdaBoost	One-vs-Remaining
1	0	0	0.989	0.996	0.99
2	0	0	0	0.207	0.204
3	0	0	0.228	0.379	0.165
4	0	0	0.98	0.843	0.98
5	0	0	0.273	0.81	0.45
6	0	0	0.925	0.968	0.968
7	0	0	0.85	0.3	1
8	0	0	0.813	0.742	0.23
9	0	0	0.167	0.216	0.166
10	0.995	0.995	0.99	0.999	0.99
11	0	0	0.806	0.914	0.78
12	0.99	0.988	1	0.999	0.998
13	0	0	0.442	0.681	0
14	0	0	0.435	0.881	0
15	0	0	0.98	0.961	0.980
16	0	0	0.947	0.983	0.962
17	0	0	0.111	0.541	0.111
18	0	0	0.968	0.978	0.986
19	0.99	0.999	1	0.999	0.999
20	0	0	0	0	0
21	0	0	0.99	0.975	0.990
22	0	0	0.258	0.926	0.953
23	0	0	0.737	0.895	0.736

[20] T. G. Dietterich and G. Bakiri, "Solving multiclass learning problems via error-correcting output codes," *Journal of Artificial Intelligence Research*, vol. 2, no. 1, pp. 263–286, 1995.

[21] KDD Cup 1999 dataset for network-based intrusion detection systems. Available on: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[22] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR*, vol. 1.

[23] W. Feng, Q. Zhang, G. Hu, and J. X. Huang, "Mining network data for intrusion detection through combining SVMs with ant colony networks," *Future Generation Computer Systems*, 2013.

[24] W. li and Z. Liu, "A method of SVM with normalization in intrusion detection," *Procedia Environmental Sciences*, vol. 11, Part A, pp. 256 – 262, 2011.

[25] H. Altwaijry and S. Algarny, "Bayesian based intrusion detection system," *Journal of King Saud University - Computer and Information Sciences*, vol. 24, no. 1, pp. 1 – 6, 2012.

[26] F. Amiri, M. R. Yousefi, C. Lucas, A. Shakery, and N. Yazdani, "Mutual information-based feature selection for intrusion detection systems," *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1184 – 1199, 2011.

[27] V. Boln-Canedo, N. Snchez-Maroo, and A. Alonso-Betanzos, "Feature selection and classification in multiple class datasets: An application to KDD cup 99 dataset," *Expert Systems with Applications*, vol. 38, no. 5, pp. 5947 – 5957, 2011.