Decision Tree Classifier For Network Intrusion Detection With GA-based Feature Selection

Gary Stein	Bing Chen	Annie S. Wu	Kien A. Hua
Computer Engineering	Computer Science	Computer Science	Computer Science
University of Central Florida	University of Central Florida	University of Central Florida	University of Central Florida
Orlando, FL 32816-2362	Orlando, FL 32816-2362	Orlando, FL 32816-2362	Orlando, FL 32816-2362
gstein@mail.ucf.edu	bchen@cs.ucf.edu	aswu@cs.ucf.edu	kienhua@cs.ucf.edu

ABSTRACT

Machine Learning techniques such as Genetic Algorithms and Decision Trees have been applied to the field of intrusion detection for more than a decade. Machine Learning techniques can learn normal and anomalous patterns from training data and generate classifiers that then are used to detect attacks on computer systems. In general, the input data to classifiers is in a high dimension feature space, but not all of features are relevant to the classes to be classified. In this paper, we use a genetic algorithm to select a subset of input features for decision tree classifiers, with a goal of increasing the detection rate and decreasing the false alarm rate in network intrusion detection. We used the KDDCUP 99 data set to train and test the decision tree classifiers. The experiments show that the resulting decision trees can have better performance than those built with all available features.

Categories and Subject Descriptors

D.1.0 Software: Programming Techniques, General

General Terms

Algorithms, Performance, Security

Keywords

Genetic Algorithm, Decision Trees, Intrusion Detection

1. INTRODUCTION

Intrusion Detection Systems (IDSs) have become a major focus of computer scientists and practitioners as computer attacks have become an increasing threat to commercial business as well as our daily lives. The computer security community has developed a variety of intrusion detection systems to prevent attacks on computer systems. There are two main categories of intrusion detection systems: anomaly detection and misuse detection. Anomaly detection systems seek to identify deviations from normal behavior models which are built from large training data sets. Misuse detection systems compare the system use behavior with signatures extracted from known attacks; a match with a high confidence is considered an attack. These two kinds of systems have their own strengths and weaknesses. The former can detect novel attacks but, in general for most such existing systems, have a high false alarm rate because it is difficult to generate practical normal behavior profiles for protected systems. The latter can detect known attacks with a very high accuracy via pattern matching on known signatures, but cannot detect novel attacks because their signatures are not yet available for pattern matching. In this paper, we only consider misuse detection systems.

Machine Learning techniques have recently been extensively applied to intrusion detection. Example approaches include decision trees [1][14][15][17][21][27], Genetic Algorithm and Genetic Programming [5][6][16][19][24], naive Bayes [1][26], kNN [18] and neural networks [4][8]. A key problem is how to choose the features (attributes) of the input training data on which learning will take place. Since not every feature of the training data may be relevant to the detection task and, in the worse case, irrelevant features may introduce noise and redundancy into the design of classifiers, choosing a good subset of features will be critical to improve the performance of classifiers [20].

Punch et al. [22], Pei et al. [20] and Huang et al. [10] address the problem of feature selection and extraction by using genetic algorithm to find an optimal (or nearly optimal) weighting of features for k-Nearest Neighbor classifiers. Huang et al. [10] apply a GA to find an optimal subset of features for a Bayes

classifier and a linear regression classifier. Gartner et al. [7] use support vector machines to find optimal feature weights for a Bayes classifier. To the best of our knowledge, combining a genetic algorithm with decision tree classifiers has not been tried for intrusion detection. In this paper, we use a genetic algorithm to find an optimal subset of features for decision tree classifiers based on the KDDCUP 99 data set with respect to the characteristics of four categories of attack: Probe, DOS, U2R and R2L.

2. RELATED WORK

In 1999, the KDD conference hosted a classifier learning contest [11], in which the learning task was to build a predictive model to differentiate attacks and normal connections. Contestants trained and tested their classifiers on an intrusion dataset provided by MIT Lincoln Labs. Each record of this dataset has 41 features consisting of three categories: (1) Basic features of individual TCP connections; (2) Content features within a connection; (3) Traffic features computed using a two-second time window. The results of the contest were based on the performance of the classifier over a testing data set of 311029 cases. Surprisingly, the top three classifiers were all decision tree classifiers [21][15][27]. These results show the capability of learning and classification of decision trees. Amor et al [1] retried the above task with naïve Bayes and decision tree classifiers. They conclude that the naïve Bayes classifier is competitive and required less training time than the decision tree classifier, although the latter had slightly better performance. All these work above use all 41 features of the KDDCUP99 training data and testing data. We argue that not all of these features are important for classifying the four categories of attack: Probe, DOS, U2R and R2L. We attempt to find an optimal subset of features for classifying each category.

Sung et al. [25] use Support Vector Machines (SVMs) and Neural Networks to identify important features for 1998 DARPA Intrusion Detection data. They delete one feature at a time and build SVMs and Neural Networks using the remaining 40 features. The importance of this deleted feature depends on training time, testing time and the accuracy for SVMs or overall accuracy, false positive rate and false negative rate for Neural Networks. The same evaluation process is done for each feature. Features are ranked according to their importance. They conclude that SVMs and neural network classifiers using only important features can achieve better or comparable performance than classifiers that use all features.

The most related work to ours is done by Huang, Pei and Goodman [10], where the general problem of GA optimized feature selection and extraction is addressed. In their paper, Huang, et al. apply a GA to optimize the feature weights of a kNN classifier and choose optimal subset of features for a Bayesian classifier and a linear regression classifier. The optimization framework of their work is based on the wrapper model [12][13]. The wrapper model consists of a search component and an evaluation component. The search component generates a parameter settings and feeds them to the evaluation component. The evaluation component then evaluates these parameter settings by induction algorithms over training and evaluation datasets. The results of the evaluation feed back to the search component that in turn generates new parameter settings. This iterative search process continues until the preset classification performance is achieved or the maximum number of loops has been reached. Experiments in [10] show that the performance of all these three classifiers with feature weighing or selection by a GA is better than that of the same classifiers without a GA. They conclude that performance gain is completely dependent on what kind of classifier is used over what type of data set.

3. INTRODUCTION TO DECISION TREES AND GENETIC ALGORITHM

3.1. Decision Trees

The decision tree classifier by Quinlan [23] is one of most well-known machine learning techniques. A decision tree is made of decision nodes and leaf nodes. Each decision node corresponds to a test X over a single attribute of the input data and has a number of branches, each of which handles an outcome of the test X. Each leaf node represents a class that is the result of decision for a case.

The process of constructing a decision tree is basically a divide-and-conquer process [23]. A set T of training data consists of k classes ($C_1, C_2, \ldots, C_k$). If T only consists of cases of one single class, T will be a leaf. If T contains no case, T is a leaf and the associated class with this leaf will be assigned with the major class of its parent node (this is the choice of C4.5). If T contains cases of mixed classes (i.e. more than one class), a test based on some attribute a_i of the training data will be carried and T will be split into n subsets ($T_1, T_2, \ldots, T_n$), where n is the number of outcomes of the test over attribute a_i. The same process of constructing decision tree is recursively performed over each T_j, where $1 \le j \le n$, until every subset belongs to a single class.

The problem here is how to choose the best attribute for each decision node during construction of the decision tree. The criterion that C4.5 chooses is Gain Ratio Criterion. The basic idea of this criterion is to, at each splitting step, choose an attribute which provides the maximum information gain while reducing the bias in favor of tests with many outcomes by normalization.

Once a decision tree is built, it can be used to classify testing data that has the same features as the training data. Starting from the root node of decision tree, the test is carried out on the same

attribute of the testing case as the root node represents. The decision process takes the branch whose condition is satisfied by the value of tested attribute. This branch leads the decision process to a child of the root node. The same process is recursively executed until a leaf node is reached. The leaf node is associated with a class that is assigned to the test case.

3.2 Genetic Algorithm

Genetic Algorithms (GAs) [9] have been successfully applied to solve search and optimization problems. The basic idea of a GA is to search a hypothesis space to find the best hypothesis. A pool of initial hypotheses called a population is randomly generated and each hypothesis is evaluated with a fitness function. Hypotheses with greater fitness have higher probability of being chosen to create the next generation. Some fraction of the best hypotheses may be retrained into the next generation, the rest undergo genetic operations such as crossover and mutation to generate new hypotheses. The size of a population is the same for all generations in our implementation. This process is iterated until either a predefined fitness criterion is met or the preset maximum number of generations is reached.

A GA generally has four components. A population of individuals where each individual in the population represents a possible solution. A fitness function which is an evaluation function by which we can tell if an individual is a good solution or not. A selection function which decides how to pick good individuals from the current population for creating the next generation. Genetic operators such as crossover and mutation which explore new regions of search space while keeping some of the current information at the same time.

The following is a typical GA procedure:

```
Procedure GA
    Begin
        Initialize population;
        Evaluate population members;
        While termination condition not satisfied do
        Begin
            Select parents from current population;
            Apply genetic operators to selected parents;
            Evaluate offspring;
            Set offspring equal to current population;
        End
    End
```

4. GA-BASED FEATURE SELECTION FOR DECISION TREES

Our proposed GA-based Feature Selection algorithm is based on the wrapper model [12][13] as discussed in section 2. In our adapted algorithm, the search component is a GA and the evaluation component is a decision tree. A detailed description of this algorithm is shown in Figure 1. The initial population is randomly generated. Every individual of the population has 41 genes, each of which represents a feature of the input data and can be assigned to 1 or 0. 1 means the represented feature is used during constructing decision trees; 0 means it is not used. As a result, each individual in the population represents a choice of available features. For each individual in the current population, a decision tree is built using the C4.5 program [23]. This resulting decision tree is then tested over nine validation data sets, which generate nine classification error rates. The fitness of this individual is the aggregate total of these classification error rates. The lower the classification error rate, the better the fitness of the individual.

Once the fitness values of all individuals of the current population have been computed, the GA begins to generate next generation as follows:

(1) Choose individuals according to Rank Selection method [2].
(2) Use two point crossover to exchange genes between parents to create offspring.
(3) Perform a bit level mutation to each offspring.
(4) Keep two elite parents and replace all other individuals of current population with offspring.

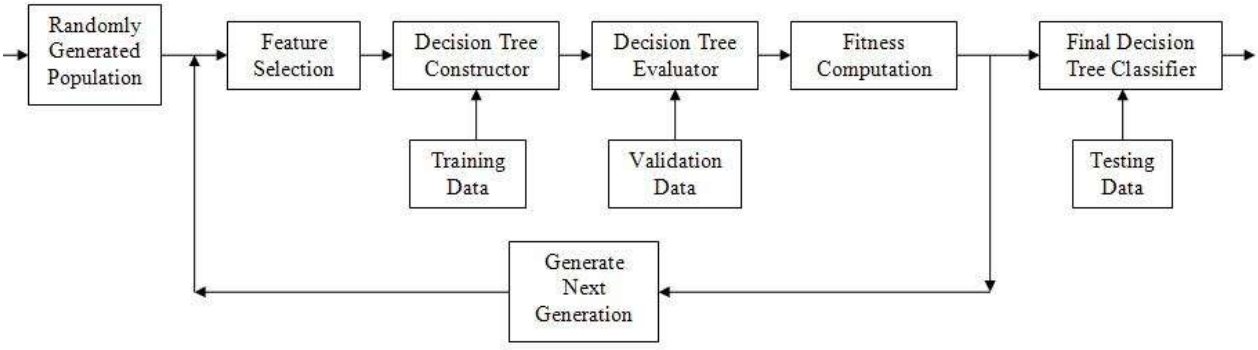

Figure 1: GA/Decision Tree Hybrid

The procedure above is iteratively executed until the maximum number of generations (100) is reached. Finally, the best individual of the last generation is chosen to build the final decision tree classifier, which is tested on the test data set.

5. EXPERIMENTS AND ANALYSIS

The main purpose of this work is to see whether the GA/Decision Tree hybrid could produce a better classification of attacks than the current best performer of Decision Tree alone. We use the 10% of the KDDCUP99 training data (489843 cases) and full testing data (311029 cases) for our experiments. We split the training data and testing data into four smaller training data sets and testing data sets according to four attack categories (Probe, DOS, R2L and U2R). For example, the training data set and test data set for DOS include all DOS attacks and all normal cases in the original training and test data. For each attack category, we split its training data into ten separate files of equal size. One is selected as the training data set and the rest as validation sets. We run the experiments for all four attack categories and build a decision tree for each category.

Each GA run is initialized with a randomly generated seed. Each gene on an individual determines whether or not a feature of the data is used in the creation of the decision tree. The fitness is the sum of the validation error rates. The test data, which is completely separate from the training and validation data, is used for the testing of the final decision tree. As part of the standard GA process, those individuals with the best fitness produce offspring and the process continues for 100 generations. In order to save computational time, we use parallel computing for our experiment. We run this experiment 20 times for each attack category.

The results of experiments for all four categories are compiled in Table 1. The "Decision Tree" column represents detection error rate of decision trees using all features on testing data set. The "Hybrid (Avg)" column represents the average of detection error rates of decision trees using features selected by top individuals

of 20 runs. The "Hybrid (Best)" column represents the best of top individuals of 20 runs that generates the least testing error rate for each category. As seen in Table 1, the average Hybrid results are typically better than those of Decision Trees. The Hybrid algorithm is able to optimize the parameters to, on average, produce better results. Upon looking at the Hybrid (best) column, however, the GA did not always converge to the same place in every instance. The GA made drastic improvements in some of the categories. For example, performance gain on Probe is 23% on the average and 60% for the best. Performance improvement on R2L and U2R are limited, however, this may be because the proportions of R2L and U2R attacks are very low in the training data, but much higher in the testing data, which include some new R2L and U2R attacks.

Table 1: Error Rate for Top Individual at the Last Generation of 20 Runs

Categories	Decision Tree (%)	Hybrid (Avg %)	Hybrid (Best %)
DOS	2.321258	2.222582	2.197655
Probe	2.166494	1.670193	0.869377
R2L	19.979205	19.945019	19.628280
U2R	0.095610	0.100885	0.089016

In order to see how genes evolve over generations, we present detail experimental results for DOS. In Figure 2, we show the change of the detection error rate over 100 generations on the validation data sets (left side) and the testing data set (right side). The value of the error rate of each generation in Figure 2 is the average of error rates of the top individuals of all 20 runs. It can be seen that the hybrid always produces better results than the

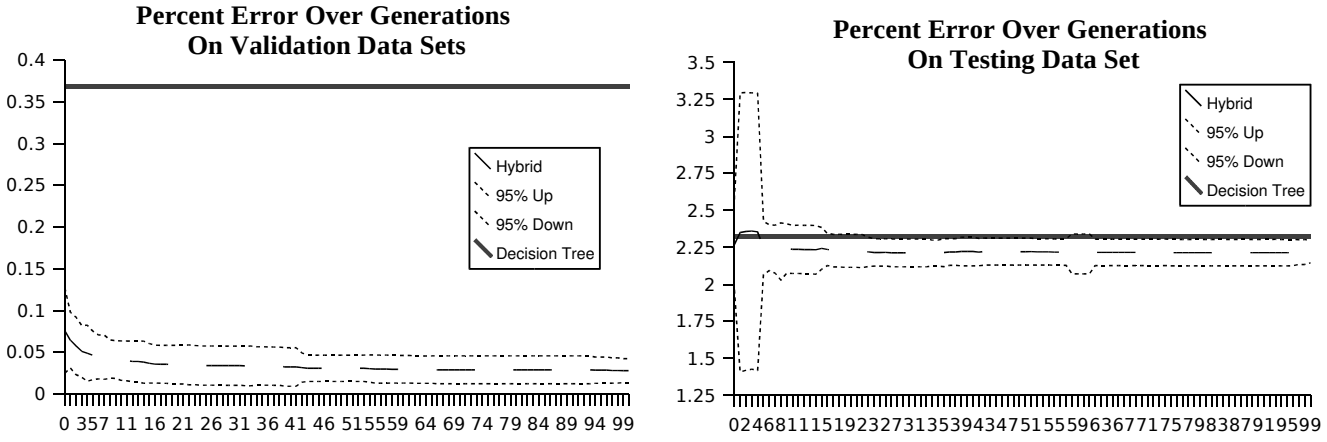

Figure 2: Sum of Percent Error Rates for Validation Data Sets (left) and Average Percent Error for Testing Data Set (right) for each Generation Averaged over 20 Runs

(Bold Line: Decision Tree, Fine Dashed Line: Hybrid, and Ultra Dashed Line: 95% confidence interval)

Table 2: Gene Frequency of Top Individual for Selected Generations over 20 Runs for the DOS Category

Attribute	Gen 0	Gen 25	Gen 50	Gen 75	Gen 100
duration	0.55	0.45	0.45	0.40	0.50
protocol_type	1.00	1.00	1.00	1.00	1.00
service	0.25	0.10	0.05	0.00	0.00
flag	0.55	0.65	0.50	0.50	0.45
src_bytes	1.00	1.00	1.00	1.00	1.00
dst_bytes	0.60	0.80	0.80	0.80	0.90
land	0.35	0.40	0.45	0.45	0.40
wrong_fragment	0.70	0.95	0.95	1.00	1.00
urgent	0.30	0.45	0.40	0.50	0.40
hot	0.45	0.30	0.25	0.15	0.20
num_failed_logins	0.55	0.30	0.40	0.40	0.45
logged_in	0.50	0.35	0.25	0.20	0.25
num_compromised	0.45	0.40	0.35	0.25	0.20
root_shell	0.45	0.35	0.50	0.50	0.60
su_attempted	0.25	0.45	0.50	0.35	0.30
num_root	0.40	0.45	0.40	0.40	0.25
num_file_creations	0.45	0.40	0.60	0.55	0.50
num_shells	0.40	0.45	0.45	0.35	0.40
num_access_files	0.45	0.45	0.50	0.40	0.45
num_outbound_cmds	0.50	0.55	0.65	0.60	0.70
is_host_login	0.50	0.60	0.55	0.55	0.65
is_guest_login	0.60	0.35	0.40	0.40	0.35
count	0.30	0.00	0.00	0.00	0.00
srv_count	0.15	0.10	0.10	0.10	0.00
serror_rate	0.70	0.80	0.85	0.80	0.80
srv_serror_rate	0.40	0.45	0.50	0.55	0.50
rerror_rate	0.65	0.70	0.60	0.50	0.40
srv_rerror_rate	0.55	0.45	0.50	0.50	0.45
same_srv_rate	0.50	0.45	0.35	0.30	0.35
diff_srv_rate	0.55	0.75	0.75	0.85	0.90
s_diff_h_rate	0.20	0.50	0.40	0.45	0.55
d_h_count	0.65	0.15	0.10	0.10	0.05

decision tree alone on the validation data set (left side). A 95% confidence interval is constructed around the average value of the 20 runs (2 standard deviations) and that is also well below the value generated by decision trees (Bold Line). A similar trend is seen in the graph for the testing data set (right side). The average value generated by the hybrid is able to improve beyond the value by decision trees toward the end of the run, when the variance decreases and the GA algorithm converges. The reason for the major increase in performance for the validation set versus the testing set can be attributed to the fact that the validation values are in the feedback loop of the wrapper model. The hybrid is able to create better decision trees with the training set because the error rate of the validation sets was the fitness function. This creates a bias toward the validation data. In addition, the testing set error was much higher than validation set error because the KDDCUP99 test dataset introduces never before seen attacks which are difficult to detect in the context of misuse detection systems.

For further analysis of how the GA is producing better results, the frequency of gene usage of the top individuals is examined for the DOS category. The meaning and the full name of each feature is given in [11]. The genes of the best individual in each generation are tracked for the 20 runs. The purpose is to discover those genes of importance that have a frequency that is either above or below some threshold away from the value of 0.50. A 0.50 frequency value means that, probabilistically, it does not matter whether that feature is on or off. As seen in Table 2, the 0^{th} generation contains the most uniform distribution of gene frequency. However, it still can be seen that the top selected individual already contains some of the basic traits of important genes (protocol_type, src_bytes). As the generations progress, the important and unimportant genes begin to travel to their respective extremes. A low frequency indicates that a particular gene is not important while a high frequency indicates that a gene is important. For example, some genes (service, hot, d_h_count, same_srv_rate) are weeded out over time and progress toward zero while others (dst_bytes, wrong_fragment, diff_srv_rate) are strengthened and increase towards one. Using some unimportant features might lead the decision tree to take the "easy way" to partition data that maximized the information gain; however, it did not create an intelligent partitioning decision. The GA portion of the algorithm was able to eliminate the unimportant features and identify those features that are necessary for effective classification.

6. CONCLUSION

The genetic algorithm and decision tree hybrid was able to outperform the decision tree algorithm without feature selection. We believe that this improvement is due to the fact that the hybrid approach is able to focus on relevant features and eliminate unnecessary or distracting features. This initial filtering is able to improve the classification abilities of the decision tree. The algorithm does take longer to execute than the standard decision tree; however, its non-deterministic process is able to make better decision trees. The training process needs only to be done once. The classification process takes the same amount of time for the hybrid and non-hybrid systems.

7. FUTURE WORK

The hybrid GA /decision tree algorithm needs to be tested more in depth for its true potential. A forest of decision trees will be constructed from the combination of four final decision trees, each for one major attack category. The final decision will be made through a voting algorithm. We will then compare the overall classification ability of the hybrid algorithm with other machine learning algorithms in the literature.

8. REFERENCES

[1] Amor, N. B., Benferhat, S., and Elouedi, Z. Naive Bayes vs decision trees in intrusion detection systems. In *Proc. 2004 ACM Symp. on Applied Computing*. pp. 420-424, 2004.

[2] Baker, J.E. Adaptive selection methods for genetic

algorithms. In *Proc. 1st Int'l Conf. On Genetic Algorithms.* Pg 101-111, 1985.

[3] Balthrop, J., Esponda, F., Forrest, S., and Glickman, M. Coverage and generalization in an artificial immune system. In *Proc. Genetic and Evolutionary Computation Conference. (GECCO)*, pp. 3-10, 2002.

[4] Botha, M., Solms, R. V., Perry, K., Loubser, E., and Yamoyany, G. The utilization of artificial intelligence in a hybrid intrusion detection system. In *Proceedings of SAICSIT 2002*, pp. 149–155, 2002.

[5] Crosbie, M., and Spafford, G. Applying genetic programming to intrusion detection. In *Proc.1995 AAAI Symposium on Genetic Programming*, pp. 1-8.

[6] Dasgupta, D., and Gonzalez, F. A. An intelligent decision support system for intrusion detection and response. In *Proc. Int'l Workshop on Mathematical Methods, Models and Arch. For Computer Networks Security,* pp. 1-14, 2001.

[7] Gartner, T., and Flach, P. A. WBCsvm: Weighted Bayesian Classification based on support vector machine. In *Proc. 18th International Conference on Machine Learning* (ICML-2001), pp. 156-161.

[8] Ghosh, A., and Schwartzbard, A. A study in using neural networks for anomaly and misuse Detection. *8th USENIX Security Symposium*, pp. 141-151, 1999.

[9] Holland, J. H. (1975). *Adaptation in natural and artificial systems*. University of Michigan Press (reprinted in 1992 by MIT Press, Cambridge, MA).

[10] Huang, Z., Pei, M., Goodman, E., Huang, Y., and Li, G. Genetic algorithm optimized feature transformation: a comparison with different classifiers. In *Proc. GECCO 2003*, pp. 2121-2133.

[11] KDDCUP 1999 http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

[12] Kohavi, R., and John, G. Automatic parameter selection by minimizing estimated error. In *Proc. Machine Learning (ICML-95)*, 1995.

[13] Kohavi, R., and John, G. The wrapper approach. In Motoda, H., and Liu, H. (eds.), *Feature Extraction, Construction and Selection: A Data Mining Perspective*. Kluwer Academic Publishers, July 1998.

[14] Kruegel, C., and Toth, T. Using decision trees to improve signature-based intrusion detection. In *Proc. Int'l Symp. Recent Advances in Intrusion Detection*, 2003.

[15] Levin, I. KDD-99 classifier learning contest LLSoft's results overview. *SIGKDD Explorations, 2000 ACM SIGKDD. 1* (2), pp. 67-75. January 2000.

[16] Li, W. Using Genetic Algorithm for network intrusion detection. In *Proc. United States Department of Energy Cyber Security Group 2004 Training Conference, Kansas City, Kansas, May 24-27, 2004.*

[17] Li, X., and Ye, N. Decision tree classifier for computer intrusion detection. *Journal of Parallel and Distributed Computing Practices, 4*(2), pp. 179-190, 2001.

[18] Liao, Y., and Vemuri, V. R. Using text categorization techniques for intrusion detection. In *Proc. of 11th USENIX Security Symposium*, San Francisco, California, USA, August 5-9, 2002.

[19] Lu, W., and Traore, I. Detecting new forms of network intrusion using genetic programming. *Computational Intelligence, 20*(3), 2004, pp. 475–494.

[20] Pei, M., Goodman, E. D., and Punch, W. F. Feature extraction using genetic algorithms. In *Proc. Int'l Symp. on Intelligent Data Engineering and Learning'98*. pp. 371-384.

[21] Pfahringer, B. Winning the KDD99 Classification Cup: Bagged boosting. *SIGKDD Explorations, 2000 ACM SIGKDD. 1*(2), pp. 65-66, January 2000.

[22] Punch, W.F., Goodman, E.D., Pei, M., Chia-Shun, L., Hovland, P., and Enbody, R. Further research on feature selection and classification using genetic algorithms. In *Proc. 5th Int'l Conf. Genetic Algorithms*. pp. 557, July 1993.

[23] Quinlan, J. R. (1993). *C4.5, Programs for Machine Learning*. Morgan Kaufmann San Mateo Ca, 1993.

[24] Sinclair, C., Pierce, L., and Matzner, S. An application of machine learning to network intrusion detection. In *Proc. 1999 Ann. Comp. Security Application Conf.*, pp 371-377.

[25] Sung, A. H., and Mukkamala, S. Identifying important features for intrusion detection using support vector machines and neural networks. In P*roceedings of the 2003 Symposium on Applications and the Internet*. pp. 209- 216.

[26] Valdes, A., Skinner K. Adaptive model-based monitoring for cyber attack detection. In *Proc. of Recent Advances in Intrusion Detection*. pp. 80-92, 2000.

[27] Vladimir, M., Alexei, V., and Ivan, S. The MP13 approach to the KDD'99 classifier learning contest. *SIGKDD Explorations, 2000 ACM SIGKDD. 1*(2), pp. 76-77. January 2000.