Indian Journal of Science and Technology, *Vol 7(6), 781–789, June 2014*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Data Mining based Hybrid Intrusion Detection System

Chandrashekhar Azad* **and Vijay Kumar Jha**

Dept. of Information Technology, Birla Institute of Technology, Mesra, Ranchi, Jhrakhand, India;
csazad@bitmesra.ac.in, vkjha@bitmesra.ac.in

Abstract

An intrusion detection system is proposed using Decision Table/Naïve Bayes (DTNB). The Proposed system uses a hybrid classifier DTNB that is used to identify possible intrusions. The system is trained using a subset of the NSL KDD Cup dataset. The trained model is then tested using a subset of NSL KDD Cup dataset. The DTNB hybrid classifier is able to detect intrusion with a superior detection rate.

Keywords: Anomaly Detection, DTNB, IDS, Misuse Detection

1. Introduction

Mankind has become more technology dependent and expects World Wide Web to handle daily requirements like newspapers, marketing, online transactions etc. The integrity, confidentiality and availability of all these web based systems are to be protected against a number of threats. Hackers and terrorists too have the purpose and capability to carry out attacks on communication system. Thus, the field of information security has become most important for safety and security beside the need based. The rapid development of electronic data processing requires safe information and security systems through the use of authentication, encryption, firewalls, intrusion detection systems, and other hardware and software solutions. Vulnerabilities in computer system such as software bugs are often manipulated by malicious users. One commonly used defense measure against such malicious attacks in the Internet is Intrusion Detection Systems (IDS). Due to increasing incidents of cyber-attacks, building effective IDS are essential for protecting web based application. IDS has emerged as a significant field of research, because it is not theoretically possible to set up a system with no vulnerabilities. One main difficulty in IDS is that we have to find out the mask attacks from the large amount of routine communication activities.

2. Intrusion Detection System

Intrusion detection is hardware or software system[1] that collects information from computer network analyzes it and finds whether there exist any abnormal activity, which violates the integrity, availability and confidentiality of the computer network. Intrusion detection system monitors communication activity and reduces possible attacks. IDS are the most important part of the security infrastructure for the local network connected to the World Wide Web or local network itself. IDS can be used to monitor unauthorized network activities, particularly IDS uses network traffic to detect, identify and track the intruder. The main goal of IDS is to alarm the network administrator if any suspicious activity happening. Mainly IDS can be classified in to two categories anomaly detection and misuse detection (signature detection)[2]. Signature based IDS references a database of previous attack signatures and known system vulnerabilities. The meaning of word signature, when we talk about IDS is recorded evidence of an attack. Each attack leaves a footprint behind (e.g., login

*Author for correspondence

failed, failed attempt to run an application, file access etc.). These footprints are called signatures and can be used to identify and prevent the same attacks in the future. Based on these signatures Signature based IDS identify intrusion attempts. Anomaly based IDS references a learned pattern of normal system activity to identify active intrusion attempts. Deviations from this pattern cause an alarm to be triggered.

Data mining plays a key role in the field of security and a number of research has been carried out viz. Ye et al.[2] in Host based intrusion detection system, Wang et al.[3] in PCA in computer security, Xiang et al.[4] in Multilevel tree classifier for intrusion detection system, Mabu s et al.[5] in fuzzy class association rule mining using genetic programming, Pastrana et al.[6] in evaluating classification algorithm for intrusion detection system MANET, Perira et al.[7] in open path forest framework, Koc et al.[9] and Sindhu et al.[8] in Decision tree based light weight intrusion detection using a wrapper approach, hidden naïve bayes multiclass classifier, Altwaijry et al.[10] in baysian based IDS Mukherjee et al.[11] in naïve bayes classifier with feature reduction, Hussain et al.[12] in hybrid IDS using snort with Naïve Bayes to detect attacks.

2.1 Decision Table

Given a training sample containing labeled instances, an induction algorithm builds a hypothesis in some representation. The representation we investigate here is a decision table with a default rule mapping to the majority class, which we abbreviate as DTM. A DTM[13] has two components:

1. A schema, which is set of features.
2. A body which is a multiset of labeled instances? Each instance consists of a value for each of the features in the schema and a value for the label.

Given a unlabeled instances U, the label assigned to the instance by a DTM classifier is computed as follows. Let L be the set of labeled instances in the DTM exactly matching the given instance I, where only the features in the schema are required to match and all other features are ignored . If L=0, returns the majority class in the DTM , otherwise return the majority class in L. Unknown values are treated as distinct values in matching process.

2.2 Naive Bayes

A Naive Bayes classifier is a simple probabilistic classifier based on applying Bayes theorem (from Bayesian statistics) with strong (naive) independence assumptions[14–15]. A more descriptive term for the underlying probability model would be 'independent feature model'. In simple terms, a Naive Bayes classifier assumes that the presence or absence of a particular feature of a class (i.e. attribute) is unrelated to the presence (or absence) of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 4 inches in diameter. Even if these features depend on each other or upon the existence of the other features, a Naive Bayes classifier considers all of these properties to independently contribute to the probability that this fruit is an apple. The advantage of the Naive Bayes classifier is that it only requires a small amount of training data to estimate the means and variances of the variables necessary for classification. Because independent variables are assumed, only the variances of the variables for each label need to be determined and not the entire covariance matrix.

The naive Bayesian classifier works as follows[15]:

1. Let D be a training set of tuples and their associated class labels. As usual, each tuple is represented by an n-dimensional attribute vector = $(x_1, x_2, \ldots .. x_n)$ depicting n measurements made on the tuple from n attributes, respectively $A_1, A_2, \ldots \ldots, A_n$.
2. Suppose that there are m classes $C_1, C_2, \ldots \ldots, C_m$. Given a tuple, X, the classifier will predict that X belongs to the class having the highest posterior probability, conditioned on X. That is, the naïve Bayesian classifier predicts that tuple X belongs to the class C_i if and only if

$$P\left(\frac{C_i}{X}\right) > \left(\frac{C_j}{X}\right) \qquad \text{for } 1 \le j \le m, \quad j \ne i$$

   Thus, we maximize $P\left(\frac{C_i}{X}\right)$. The class C_i for which $P\left(\frac{C_i}{X}\right)$ is maximized is called the maximum posteriori hypothesis. By Bayes' theorem
3. As P(X) is the same for all classes, only $P(X|C_i)P(C_1)$ need be maximized. If the class a priori probabilities, $P(C_i)$, are not known, then it is commonly assumed that the classes are equally likely, that is, $P(C_1) = P(C_2) = \ldots = P(C_m)$, and we would therefore maximize $P(X|C_i)$. Otherwise we maximize $P(X|C_i)P(C_1)$. Note that the class a priori probabilities may be estimated by $P(C_i) = |C_{(i,D)}| / |D|$ where the $|C_{(i,D)}|$ is the number of training tuples of the class C_i in D.
4. Given data sets with many attributes, it would be computationally expensive to compute $P(X|C_i)$. in order to reduce computation in evaluating $P(X|C_i)P(C_1)$, the

naive assumption of class conditional independence is made. This presumes that the values of the attributes are conditionally independent of one another, given the class label of the sample. Mathematically this means that

$$P\left(\frac{X}{C_i}\right)=\prod_{k=1}^{n}P\left(\frac{x_k}{C_1}\right)$$
$$P\left(\frac{x_1}{C_i}\right)\times P\left(\frac{x_2}{C_i}\right)\times\ldots\times P\left(\frac{x_n}{C_i}\right)$$

We can easily estimate the probabilities $P\left(\frac{x_1}{C_i}\right), P\left(\frac{x_2}{C_i}\right), \ldots, P\left(\frac{x_n}{C_1}\right)$ from the training set.

Here x_k is value of the attribute A_k for tuple X.

(a) If A_k is categorical, then $P\left(\frac{x_k}{C_i}\right)$ is the number of samples of class C_i in D having the value x_k for attribute A_k, divided by $|C_{i,D}|$, the number of sample of class C_i in D.

(b) If A_k is continuous valued, then we typically assume that the values have a Gaussian distribution with a mean μ and standard deviation σ defined by

$$g=(x,\mu,\sigma)=\frac{1}{\sqrt{2\pi\sigma}}\ e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$
$$\text{so that } P\left(\frac{x_k}{C_1}\right)=g(x_k,\mu c_i,\sigma c_1)$$

We need to compute μc_i and σc_i, which are the mean and standard deviation of values of attribute A_k for training samples of class C_i.

5. In order to predict the class label of X, $P(X|C_i)P(C_1)$ is evaluated for each class C_i. The classifier predicts that the class label of X is C_i if and only if it is the class that maximizes $P(X|C_i)P(C_1)$.

2.3 DTNB Hybrid Classifier[16]

A DT (decision table) stores the input data in condensed form based on a selected set of attributes and uses it as a lookup table when making predictions. Each entry in the table is associated with class probability estimates based on observed frequencies. The key to learning a DT is to select a subset of highly discriminative attributes. The standard approach is to choose a set by maximizing cross-validated performance. Cross-validation is efficient for DTs as the structure does not change when instances are added or deleted, only the class counts associated with the entries change. Similarly, cross validation for Naive Bayes (NB) is also efficient as frequency counts for discrete attributes can be updated in constant time. In our experiments we used forward selection to select attributes in stand-alone DTs because it performed significantly better than backward selection. Numeric attributes in the training data (including those to be modeled by NB) were discretized using MDL-based discretization (Fayyad & Irani 1993), with intervals learned from the training data. The algorithm for learning the combined model (DTNB) proceeds in much the same way as the one for stand-alone DTs. At each point in the search it evaluates the merit associated with splitting the attributes into two disjoint subsets: one for the DT, the other for NB. We use a forward selection, where, at each step, selected attributes are modeled by NB and the remainder by the DT, and all attributes are modeled by the DT initially. Leave-one-out cross-validated AUC is used to evaluate the quality of a split based on the probability estimates generated by the combined model. Note that AUC can easily be replaced by other performance measures. We chose AUC to enable a fair comparison to NB (and hence only used two-class datasets in our experiments). AUC was also used to select attributes for the stand-alone DT. The class probability estimates of the DT and NB must be combined to generate overall class probability estimates. Assuming $X^{\top}$ is the set of attributes in the DT and $X^{\perp}$ the one in NB, the overall class probability is computed as

$$Q(y|X)=\alpha Q_{DT}(y|X^{\top})\times\frac{Q_{NB}(y|X^{\perp})}{Q(y)},$$

where $Q_{DT}(y|X^{\top})$ and $Q_{NB}(y|X^{\perp})$ are the class probability estimates obtained from the DT and NB respectively, α is normalization constant, and Q(y) is the prior probability of the class. All probabilities are estimated using Laplace corrected observed counts.

In addition to the method described above, we also consider a variant that includes attribute selection, which can discard attributes entirely from the combined model. To this end, in each step of the forward selection, an attribute can be discarded rather than added to the NB model.

3. Dataset

KDD'99 is the mostly used data set for the anomaly detection methods, in contains around 2 million records in test data and approximately 4,900,000 records in training dataset, each of which contains 41 features and one decision attribute. NSL-KDD is a data set

suggested to solve some of the inherent problems of the KDD'99 data set[17]. NSL KDD cup dataset[18] does not include redundant records in the train set and test set. To test our IDS system we used the NSL KDD Intrusion Detection Evaluation dataset, in our experiments we used the 24999 randomly chosen records from NSL KDD dataset.

4. Architecture of Proposed IDS

4.1 Steps for Experiment

Step1: Preprocessing

- Normalization – Scaling
- Class merging
- Role setting
- Data Filtering

Step2: Feature. Selection

Step3: Model Learning

- DBTN

Step4: Evaluation of Model

Step5: Compare with supervised method

4.2 Experiments and Result

For our experiments, we use the NSL KDD intrusion data set. This data set is improved version of KDD 99 intrusion detection. Each record in the data set represents a connection between source and destination IP addresses, starting and ending at some well-defined times with a well-defined protocol. Further, every record is represented by 41 different features and a special decision attribute. Each record represents a separate connection and is hence considered to be independent of any other record. The training data is either labeled as normal or as one of the 22 different kinds of attacks similarly, the test data is also labeled as either normal or as one of the 22 different kinds of attacks. In our experiment NSL KDD data set is used to train the model, by applying the DTNB hybrid classifier. After model training we applied NSL KDD test data set to test the model, our testing dataset contains 24999 randomly chosen records. On testing data set DTNB model gave 97.1399 % instance correctly classified and 2.8601 % instance incorrectly classified. Table 1 shows the detailed accuracy of the framework class. Table 2 shows the performance of the DTNB based proposed framework, and the Table 3 shows the confusion matrix generated by the model.

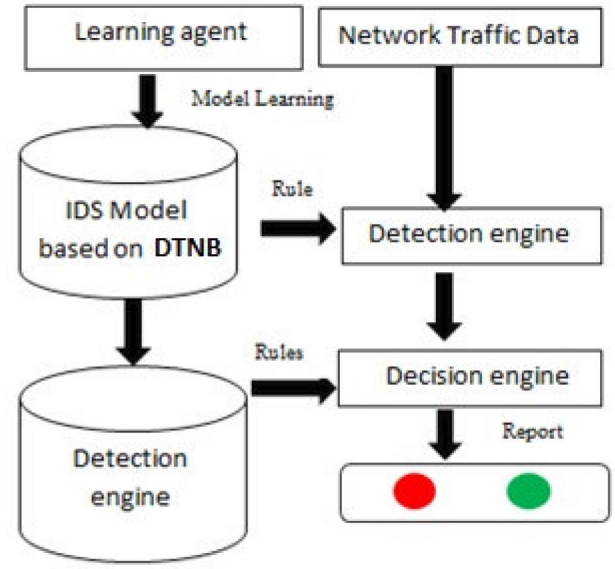

Figure 1. IDS framework.

4.3 Visualization of the Performance of Model thrh Confusion Matrix

A confusion matrix is a specific rectangular layout that allows visualization of the performance of a model or algorithm. Each column of the matrix represents the instances in a predicted class, while each row represents the instances in an actual class.

		Predicted	
Actual	class	True	False
	True	TP	FN
	False	FP	TN

Table 1. Testing result of NSL KDD test datset

Parameter	Result
Correctly Classified Instances	97.1399 %
Incorrectly Classified Instances	2.8601 %
Kappa statistic	0.9531
Mean absolute error	0.0047
Root mean squared error	0.0464
Relative absolute error	7.8043 %
Root relative squared error	26.8136 %
Coverage of cases (0.95 level)	99.12 %
Mean rel. region size (0.95 level)	6.5143 %

Table 2. Dataset Description

Attribute	Description	Type
duration	duration (number of seconds) of the connection	Numeric
protocol_type	type of the protocol, e.g. tcp, udp, etc.	Nominal
service	network service on the destination, e.g. http, telnet, etc.	Nominal
src_bytes	number of data bytes from source to destination	Numeric
dst_bytes	number of data bytes from destination to source	Numeric
flag	Normal or error flag status of the connection	Nominal
land	1 if connection is from/to the same host/port; 0 otherwise	Numeric
wrong_fragment	number of ``wrong" fragments	Numeric
urgent	number of urgent packets	Numeric
hot	number of ``hot" indicators	Numeric
num_failed_logins	number of failed login attempts	Numeric
logged_in	1 if successfully logged in; 0 otherwise	Numeric
num_compromised	number of ``compromised" conditions	Numeric
root_shell	1 if root shell is obtained; 0 otherwise	Numeric
su_attempted	1 if ``su root" command attempted; 0 otherwise	Numeric
num_root	number of ``root" accesses	Numeric
num_file_creations	number of file creation operations	Numeric
num_shells	number of logins of normal users	Numeric
num_access_files	number of operations on access control files	Numeric
num_outbound_cmds	number of outbound commands in an ftp session	Numeric
is_host_login	1 if the login belongs to the ``hot" list; 0 otherwise	numeric
is_guest_login	1 if the login is a ``guest"login; 0 otherwise	Numeric
count	number of connections to the same host as the current connection in the past two seconds	Numeric
srv_count	sum of connections to the same destination port number	Numeric
serror_rate	% of connections that have ``SYN" errors	Numeric
rerror_rate	% of connections that have ``REJ" errors	Numeric
same_srv_rate	% of connections to the same service	Numeric
diff_srv_rate	% of connections to different services	Numeric
srv_serror_rate	% of connections that have ``SYN" errors	Numeric
srv_rerror_rate	% of connections that have ``REJ" errors	Numeric
srv_diff_host_rate	% of connections to different hosts	Numeric
dst_host_count	sum of connections to the same destination IP address	Numeric
dst_host_srv_count	sum of connections to the same destination port number	Numeric
dst_host_same_srv_rate	% of connections that were to the same service, among the connections aggregated in dst_host_count	Numeric
dst_host_diff_srv_rate	% of connections that were to different services, among the connections aggregated in dst_host_count	Numeric
dst_host_same_src_port_rate	% of connections that were to the same source port, among the connections aggregated in dst_host_srv_count	Numeric
dst_host_srv_diff_host_rate	% of connections that were to different destination machines, among the connections aggregated in dst_host_srv_count	Numeric
dst_host_serror_rate	% of connections that have activated the flag s0, s1, s2 or s3, among the connections aggregated in dst_host_count	Numeric
dst_host_srv_serror_rate	% of connections that have activated the flag s0, s1, s2 or s3, among the connections aggregated in dst_host_srv_count	Numeric
dst_host_rerror_rate	% of connections that have activated the flag REJ, among the connections aggregated in dst_host_count	Numeric
dst_host_srv_rerror_rate	% of connections that have activated the flag REJ, among the connections aggregated in dst_host_srv_count	Numeric
class	Type of attacks	Nominal

Table 3. Detailed Accuracy by Class

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0.959	0.002	0.998	0.959	0.978	0.955	1.000	1.000	normal
0.983	0.004	0.847	0.983	0.910	0.910	0.999	0.980	portsweep
0.995	0.000	1.000	0.995	0.997	0.996	1.000	1.000	neptune
0.969	0.002	0.947	0.969	0.958	0.957	0.998	0.982	satan
0.942	0.003	0.914	0.942	0.928	0.926	0.999	0.975	ipsweep
1.000	0.000	0.769	1.000	0.870	0.877	1.000	1.000	guess_pass
1.000	0.001	0.912	1.000	0.954	0.954	1.000	1.000	back
0.000	0.000	0.000	0.000	0.000	0.000	0.998	0.016	land
0.600	0.006	0.021	0.600	0.041	0.112	0.972	0.103	imap
0.910	0.003	0.783	0.910	0.842	0.842	0.999	0.858	nmap
0.974	0.000	0.822	0.974	0.892	0.895	1.000	0.980	pod
1.000	0.000	0.989	1.000	0.994	0.994	1.000	1.000	smurf
0.000	0.000	0.000	0.000	0.000	0.000	0.807	0.000	ftp_write
0.250	0.001	0.053	0.250	0.087	0.114	0.946	0.013	rootkit
0.000	0.007	0.000	0.000	0.000	-0.001	0.952	0.004	warezmaster
0.667	0.002	0.075	0.667	0.136	0.224	0.957	0.534	buffer_overflo
0.000	0.000	0.000	0.000	0.000	0.000	1.000	1.000	loadmodule
0.000	0.000	0.000	0.000	0.000	0.000	0.876	0.001	multihop
0.000	0.000	0.000	0.000	0.000	0.000	0.993	0.013	phf
1.000	0.000	0.979	1.000	0.989	0.989	1.000	1.000	teardrop
0.971	0.001	0.986	0.971	0.978	0.965	1.000	0.995	Weighted Avg

Table 4. Confusion Matrix

a	b	c	d	e	F	g	h	i	j	k	L	m	n	o	p	q	r	s	t	classified as
12890	67	0	26	37	3	17	0	110	59	5	6	0	17	153	48	0	0	0	4	a=normal
0	577	1	4	0	0	0	0	2	1	0	0	0	0	2	0	0	0	0	0	b=portsweep
0	4	8240	7	0	0	0	0	25	0	0	0	0	0	6	0	0	0	0	0	c=neptune
4	3	0	666	0	0	1	0	0	7	3	0	0	1	2	0	0	0	0	0	d=satan
4	30	0	0	669	0	0	0	0	7	0	0	0	0	0	0	0	0	0	0	e=ipsweep
0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	f=guess_pass
0	0	0	0	0	0	196	0	0	0	0	0	0	0	0	0	0	0	0	0	g=back
0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	h=land
1	0	0	0	0	0	1	0	3	0	0	0	0	0	0	0	0	0	0	0	i=imap
0	0	0	0	26	0	0	0	1	274	0	0	0	0	0	0	0	0	0	0	j=nmap
0	0	0	0	0	0	0	0	0	1	37	0	0	0	0	0	0	0	0	0	k=pod
0	0	0	0	0	0	0	0	0	0	0	529	0	0	0	0	0	0	0	0	l=smurf
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	m=ftp_write
2	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	n=rootkit
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	o=warezmaster
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	0	0	0	p=bufferoverflo
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	q=loadmodule
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	r=multihop
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	s=phf
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	188	t=teardrop

5. Discussion

In this paper, the following parameters are used to evaluate the performance:

- Mean Absolute Error (MAE)
- Root Mean-Squared Error (RMSE)
- Correctly Classified Instances
- Incorrectly Classified Instances
- Kappa Statistic
- Relative absolute error
- Root relative squared error
- Coverage of cases (0.95 level)
- Mean rel. region size (0.95 level)

Table 5 shows the comparison of Naive Bayes and Decision Stump with the hybrid classifier used in the proposed IDS framework. Here we see that DTNB has better accuracy as compared to the Naïve bayes and decision stump.

Here we provide the graphical representation of various parameters used for performance evaluation. Figure 2 shows the comparison of correctly classified instance and incorrectly classified instance, We can see that DTNB hybrid classifier 97.1399 %, Naïve Bayes 53.4261 % and Decision Stump 83.9194 instance correctly classified. Figure 3shows the comparison of kappa, mean absolute error and root mean square error. Figure 4 shows the relative absolute error, root relative squared error, coverage of cases at 0.95 levels and mean rel. region size at 0.95 levels. Figures 5 provide the graphical representation of confusion matrix; it shows the ratio of actual class and predicted class in terms of correctly classified instance.

Table 5. Comparison of DTNB and other Models

Parameter	DTNB	Naive Bayes	decision stump
Correctly Classified Instances	97.1399 %	53.4261 %	83.9194 %
Incorrectly Classified Instances	2.8601 %	46.5739 %	16.0806 %
Kappa statistic	0.9531	0.4297	0.7099
Mean absolute error	0.0047	0.0461	0.0281
Root mean squared error	0.0464	0.2111	0.1185
Relative absolute error	7.8043 %	77.0879 %	46.9274 %
Root relative squared error	26.8136 %	122.0494 %	68.5196 %
Coverage of cases (0.95 level)	99.12 %	57.0103 %	96.4359 %
Mean rel. region size (0.95 level)	6.5143 %	5.4838 %	16.6879 %

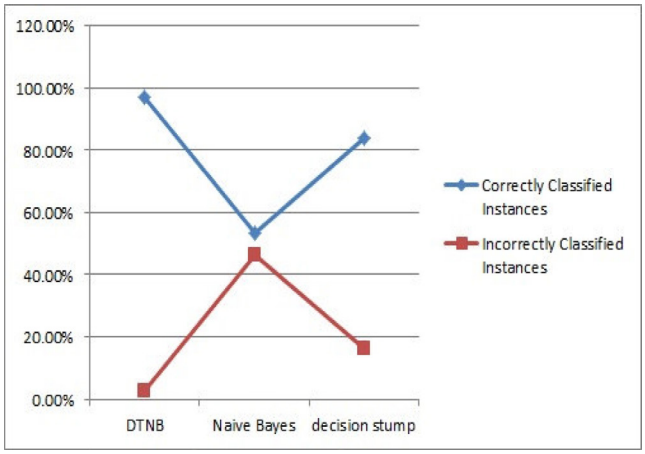

Figure 2. Comparison of classified instance.

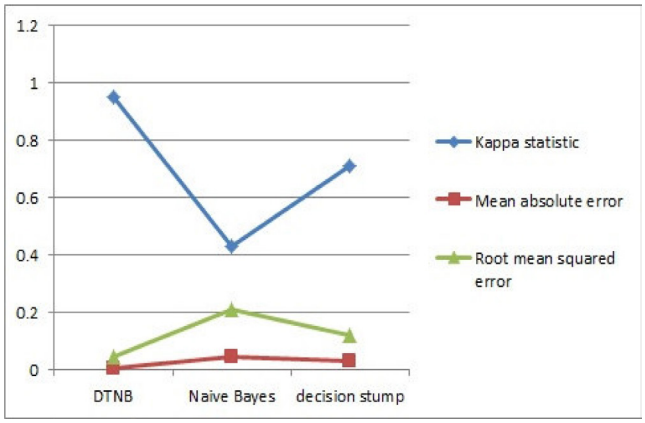

Figure 3. Comparison of Kappa, MAE, RMSE.

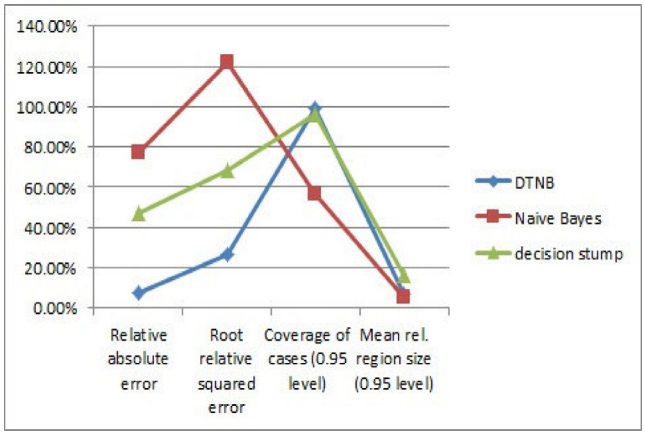

Figure 4. Comparison of RAE, RRSE, CC and MRRS.

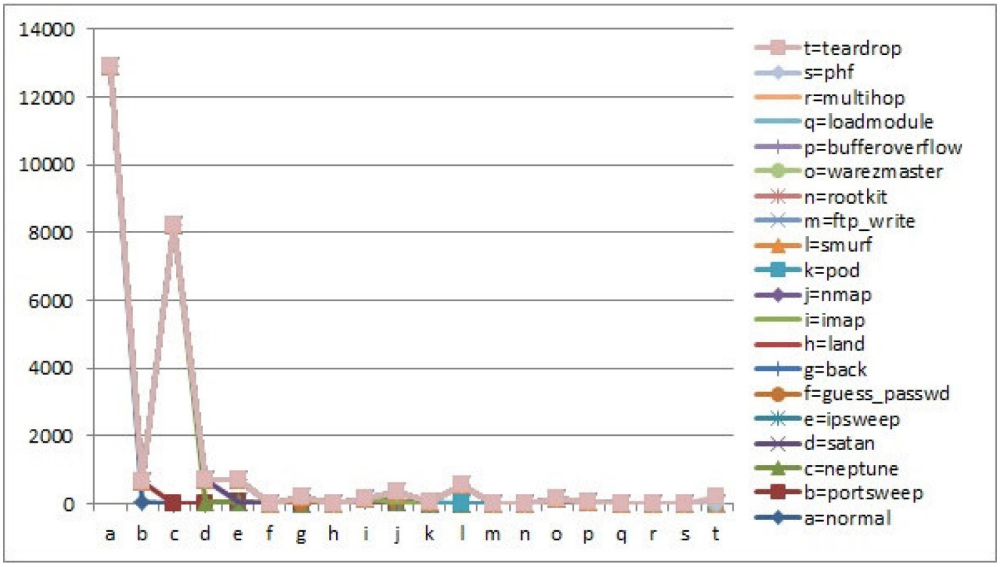

Figure 5. Graphical representation of confusion matrix.

6. Conclusion

In this paper data mining based intrusion detection framework is proposed, framework is based on the DTNB (Decision Table/ Naïve Bayes). NSL KDD dataset is used to test the effectiveness of the DTNB based model. Fundamental concept behind choosing NSL KDD dataset is , there are so many errors in the KDD data set like redundancy, Due to errors KDD dataset may give biased result, that's why here we chosen NSL KDD dataset to test the effectiveness of the DTNB model. To analyses the performance of discussed classifiers, in this paper different parameters are used, are Correctly Classified instance, Incorrectly Classified instance, Mean Absolute Error (MAE), Root Mean Square Error (RMSE) and Kappa Statistic. After analyses, the result of hybrid classifier used in the proposed framework provides better performance on behalf of following parameters correctly classified instance, incorrectly classified instance, RMSE, MAE, Time Taken and Kappa Statistic as compared to the Naïve Bayes and Decision Stump.

7. References

1. Available from: http://en.wikipedia.org/wiki/Intrusion_detection_system
2. Ye N, Emran SM, Chen Q, Vilbert S. Multivariate statistical analysis of audit trails for host-based intrusion detection. IEEE Transactions on Computers. 2002; 51.
3. Wang W, Guan X, Zhang X. A novel intrusion detection method based on principle component analysis in computer security. Springer-Verlag Berlin Heidelberg. 2004; 657–62.
4. Xiang C, Chong MY, Zhu HL. Design of multiple-level tree classifiers for intrusion detection system. Proceedings of the 2004 IEEE Conference on Cybernetics and Intelligent Systems Singapore; 2004.
5. Mabu S, Chen C, Lu N, Shimada K, Hirasawa K. An intrusion-detection model based on fuzzy class-association-rule mining using genetic network programming. IEEE Transactions on systems, MAN, and Cybernetics—Part C: Applications and Reviews. 2011; 41.
6. Pastrana S, Mitrokotsa A, Orfila A, Peris–Lopez P. Evaluation of classification algorithms for intrusion detection in MANETs. Knowledge Based Systems. 2012; 36:217–25.
7. Pereira CR, Nakamura RYM, Costa KAP, Papa JP. An optimum path forest framework for intrusion detection in computer networks. Engineering Applications of Artificial Intelligence. 2012; 25:1226–34.
8. Sindhu SSS, Geetha S, Kannan A. Decision tree based light weight intrusion detection using a wrapper approach. Expert Systems with Applications. 2012; 39:129–41.
9. Koc L, Mazzuchi TA, Sarkani S. A network intrusion detection system based on a Hidden Naïve Bayes multiclass classifier. Expert Systems with Applications. 2012; 39:13492–500.
10. Altwaijry H, Algarny S. Bayesian based intrusion detection system. Journal of King Saud University – Computer and Information Sciences. 2012; 24:1–6.

11. Mukherjee DS, Sharma N. Intrusion detection using Naive Bayes classifier with feature reduction. Procedia Technology. 2012; 4:119–28.
12. Hussein SM, Ali FHM, Kasiran Z. Evaluation effectiveness of hybrid IDS using snort with Naive Bayes to detect attacks. IEEE. 2012.
13. Kohavi R. The power of decision tables. 8th European Conference on Machine Learning. 1995. p. 174–89
14. John GH, Langley P. Estimating continuous distributions in bayesian classifiers. Eleventh Conference on Uncertainty in Artificial Intelligence. 1995; San Mateo. P. 338–45.
15. Han J, Kamber M. Data mining concepts and techniques. 2nd ed.
16. Hall M, Frank E. Combining Naive Bayes and decision tables. Proceedings of the 21st Florida Artificial Intelligence Society Conference (FLAIRS); 2008.
17. Tavallaee M, Bagheri E, Lu W, Ghorbani A. A detailed analysis of the KDD CUP 99 data set. Submitted to Second IEEE Symposium on Computational Intelligence for Security and Defense Applications (CISDA); 2009.
18. Available from: http://nsl.cs.unb.ca/NSL-KDD/