Chapter 15

Mining Complex Network Data for Adaptive Intrusion Detection

Dewan Md. Farid, Mohammad Zahidur Rahman
and Chowdhury Mofizur Rahman

Additional information is available at the end of the chapter

http://dx.doi.org/10.5772/48308

1. Introduction

Intrusion detection is the method of identifying intrusions or misuses in a computer network, which compromise the confidentiality and integrity of the network. Intrusion Detection System (IDS) is a security tool used to monitor network traffic and detect unauthorized activities in the network [23, 28, 30]. A security monitoring surveillance system, which is an intrusion detection model based on detecting anomalies in user behaviors was first introduced by James P. Anderson in 1980 [1]. After that several intrusion detection models based on statistics, Markov chains, time-series, etc proposed by Dorothy Denning in 1986. At first host-based IDS was implemented, which located in the server machine to examine the internal interfaces [35], but with the evolution of computer networks day by day focus gradually shifted toward network-based IDS [20]. Network-based IDS monitors and analyzes network traffics for detecting intrusions from internal and external intruders [26, 27, 34]. A number of data mining algorithms have been widely used by the intelligent computational researchers in the large amount of network audit data for detecting known and unknown intrusions in the last decade [3, 9, 18, 32, 33]. Even for a small network the amount of network audit data is very large that an IDS needs to examine. Use of data mining for intrusion detection aim to solve the problem of analyzing the large volumes of audit data and realizing performance optimization of detection rules.

There are many drawbacks in currently available commercial IDS, such as low and unbalanced detection rates for different types of network attacks, large number of false positives, long response time in high speed network, and redundant input attributes in intrusion detection training dataset. In general a conventional intrusion detection dataset is complex, dynamic, and composed of many different attributes. It has been successfully tested that not all the input attributes in intrusion detection training dataset may be needed for training the intrusion detection models or detecting intrusions [31]. The use of redundant attributes interfere with the correct completion of mining task, increase the complexity of detection model and computational time, because the information they added is contained in other attributes [7]. Ideally, IDS should have an intrusion detection rate of 100% along with false positive of 0%, which is really very difficult to achieve.

Applying mining algorithms for adaptive intrusion detection is the process of collecting network audit data and convert the collected audit data to the format that is suitable for mining. Finally, developing a clustering or classification model for intrusion detection, which provide decision support to intrusion management for detecting known and unknown intrusions by discovering intrusion patterns [4, 5].

2. Intrusion detection

Intrusion detection is the process of monitoring and analyzing the network traffics. It takes sensor data to gather information for detecting intrusions from internal and external networks [6], and notify the network administrator or intrusion prevention system (IPS) about the attack [19, 24]. Intrusion detection is broadly classified into three categories: misuse, anomaly, and hybrid detection model [10]. Misuse detection model detects attacks based on known attack patterns, which already stored in the database by using pattern matching of incoming network packets to the signatures of known intrusions. It begins protecting the network immediately upon installation and produces very low FP, but it requires frequently signature updates and cannot detect new intrusions. Anomaly based detection model detects deviations from normal behaviors to identify new intrusions [22]. It creates a normal profile of the network and then any action that deviated from the normal profile is treated as a possible intrusion, which produces large number of false positives. Hybrid detection model combines both misuse and anomaly detection models [2]. It makes decision based on both the normal behavior of the network and the intrusive behavior of the intruders. Table 1 shows the comparisons among misuse, anomaly, and hybrid detection models.

Characteristics	Misuse	Anomaly	Hybrid
Detection Accuracy	High (for known attacks)	Low	High
Detecting New Attacks	No	Yes	Yes
False Positives	Low	Very high	High
False Negatives	High	Low	Low
Timely Notifications	Fast	Slow	Rather Fast
Update Usage Patterns	Frequent	Not Frequent	Not Frequent

Table 1. Comparisons of Detection Models.

Detection rate (DR) and false positive (FP) are the most important parameters that are used for performance estimation of intrusion detection models [8]. Detection rate is calculated by the number of intrusions detected by the IDS divided by the total number of intrusion instances present in the intrusion dataset, and false positive is an alarm, which rose for something that is not really an attack, which are expressed be equation 1 and 2.

$$DetectionRate, DR = \frac{TotalDetectedAttacks}{TotalAttacks} * 100 \tag{1}$$

$$FalsePositive, FP = \frac{TotalMisclassifiedProcess}{TotalNormalProcess} * 100 \tag{2}$$

Also precision, recall, overall, and false alarm have been used to measure the performance of IDS [21, 25] from table 2, precision, recall, overall, and false alarm may be expressed be equation 3 to 6.

Parameters	Definition
True Positive (TP) or Detection Rate (DR)	Attack occur and alarm raised
False Positive (FP)	No attack but alarm raised
True Negative (TN)	No attack and no alarm
False Negative (FN)	Attack occur but no alarm

Table 2. Parameters for performances estimation of IDS.

$$Precision = \frac{TP}{TP + FP} \tag{3}$$

$$Recall = \frac{TP}{TP + FN} \tag{4}$$

$$Overall = \frac{TP + TN}{TP + FP + FN + TN} \tag{5}$$

$$FalseAlarm = \frac{FP + FN}{TP + FP + FN + TN} \tag{6}$$

2.1. Intrusion detection dataset

The data generated by IDS contain information about network topology, hosts and other confidential information for this reason intrusion detection dataset cannot be shared in public domain. Whereas it is not difficult task to generate a large set of intrusion detection alets by running IDS in a private or Internet-exposed network. For generating intrusion detection dataset only major challenge is to labeling the network data, because network data are unlabeled and it is not clear which attacks are false positives and which are true positives. The KDD 1999 cup benchmark intrusion detection dataset is the most popular dataset for intrusion detection research, a predictive model capable of distinguishing between intrusions and normal connections, which was first used in the 3^{rd} International Knowledge Discovery and Data Mining Tools Competition for building a network intrusion detector [29]. A simulated environment was set up by the MIT Lincoln Lab to acquire raw TCP/IP dump data for a local-area network (LAN) to compare the performance of various IDS that is the part of KDD99 dataset. Examples in KDD99 dataset represent attribute values of a class in the network data flow, and each class is labelled either normal or attack. For each network connection 41 attributes are in KDD99 dataset that have either discrete or continuous values. These attributes are divided into three groups: basic features , content features, and statistical features of network connection.

The classes in KDD99 dataset are mainly categorized into five classes: normal, denial of service (DoS), remote to user (R2L), user to root (U2R), and probing. Normal connections are generated by simulated daily user behaviours. Denial of service computes power or memory of a victim machine too busy or too full to handle legitimate requests. Remote to user is an attack that a remote user gains access of a local user or account by sending packets to a machine over a network communication. User to root is an attack that an intruder begins with the access of a normal user account and then becomes a root-user by exploiting various vulnerabilities of the system. Probing is an attack that scans a network to gather information or find known vulnerabilities. In KDD99 dataset the main four attacks are divided into 22 different attacks that tabulated in table 3 and table 4 shows the number of training and testing examples for each major class.

4 Main Attack Classes	22 Attacks Classes
Denial of Service (DoS)	back, land, neptune, pod, smurf, teardrop
Remote to User (R2L)	ftp_write, guess_passwd, imap, multihop, phf, spy, warezclient, warezmaster
User to Root (U2R)	buffer_overflow, perl, loadmodule, rootkit
Probing	ipsweep, nmap, portsweep, satan

Table 3. Different types of attacks in KDD99 Dataset.

Attack Types	Training Examples	Testing Examples
Normal	97278	60592
Denial of Service	391458	237594
Remote to User	1126	8606
User to Root	52	70
Probing	4107	4166
Total Examples	494021	311028

Table 4. Number of training and testing examples in KDD99 dataset.

3. Combining naïve Bayesian and Decision Tree

In this section, we present the hybrid learning algorithms, NBDTAID (naïve Bayesian with Decision Tree for Adaptive Intrusion Detection) [14], ACDT (Attacks Classificaton using Decision Tree) [13], and Attribute Weighting with Adaptive NBTree Algorithm [11, 15] for intrusions classification in intrusion detection problem. Presented algorithms are performed balance detections and keep FP at acceptable level for different types of network intrusions. It has been successfully tested that by combining the properties of naïve Bayesian classifier and decision tree classifier, the performance of intrusion detection classifier can be enhanced.

3.1. Adaptive intrusion classifier

Naïve Bayesian with Decision Tree for Adaptive Intrusion Detection (NBDTAID) performs balance detections and keeps FP at acceptable level in intrusion detection. NBDTAID eliminates redundant attributes and contradictory examples from training data, and addresses some mining difficulties such as handling continuous attribute, dealing with missing attribute values, and reducing noise in training data [14].

Given a training data $D = \{t_1, \cdots, t_n\}$ where $t_i = \{t_{i1}, \cdots, t_{ih}\}$ and the training data D contains the following attributes $\{A_1, A_2, \cdots, A_n\}$ and each attribute A_i contains the following attribute values $\{A_{i1}, A_{i2}, \cdots, A_{ih}\}$. The attribute values can be discrete or continuous. Also the training data D contains a set of classes $C = \{C_1, C_2, \cdots, C_m\}$. Each example in the training data D has a particular class C_j. The algorithm first searches for the multiple copies of the same example in the training data D, if found then keeps only one unique example in the training data D (suppose all attribute values of two examples are equal then the examples are similar). Then the algorithm discreties the continuous attributes in the training data D by finding each adjacent pair of continuous attribute values that are not classified into the same class value for that continuous attribute. Next the algorithm calculates the prior $P(C_j)$ and conditional $P(A_{ij}|C_j)$ probabilities in the training data D. The prior probability $P(C_j)$ for each class is estimated by counting how often each class occurs in the training data D. For each attribute A_i the number of occurrences of each attribute value A_{ij} can be counted to determine $P(A_i)$. Similarly, the conditional probability $P(A_{ij}|C_j)$ for each attribute values A_{ij} can be estimated by counting how often each attribute value occurs in the class in the training data D. Then the algorithm classifies all the examples in the training data D with these prior $P(C_j)$ and conditional $P(A_{ij}|C_j)$ probabilities. For classifying the examples, the prior and conditional probabilities are used to make the prediction. This is done by combining the effects of the different attribute values from that example. Suppose the example e_i has independent attribute values $\{A_{i1}, A_{i2}, \cdots, A_{ip}\}$, we know $P(A_{ik}|C_j)$, for each class C_j and attribute A_{ik}. We then estimate $P(e_i|C_j)$ by

$$P(e_i|C_j) = P(C_j) \prod_{k=1}^{p} P(A_{ij}|C_j) \tag{7}$$

To classify the example, the algorithm estimates the likelihood that e_i is in each class. The probability that e_i is in a class is the product of the conditional probabilities for each attribute value with prior probability for that class. The posterior probability $P(C_j|e_i)$ is then found for each class and the example classifies with the highest posterior probability for that example. After classifying all the training examples, the class value for each example in training data D updates with Maximum Likelihood (ML) of posterior probability $P(C_j|e_i)$.

$$C_j = C_i \rightarrow P_{ML}(C_j|e_i) \tag{8}$$

Then again the algorithm calculates the prior $P(C_j)$ and conditional $P(A_{ij}|C_j)$ probabilities using updated class values in the training data D, and again classifies all the examples of training data using these probabilities. If any of the training example is misclassified then the algorithm calculates the information gain for each attributes $\{A_1, A_2, \cdots, A_n\}$ in the training data D.

$$Info(D) = -\sum_{j=1}^{m} \frac{freq(C_j, D)}{|D|} log_2 \left(\frac{freq(C_j, D)}{|D|} \right) \tag{9}$$

$$Info(T) = -\sum_{i=1}^{n} \frac{|T_i|}{|T|} info(T_i) \tag{10}$$

$$InformationGain(A_i) = Info(D) - Info(T) \tag{11}$$

And chooses one of the best attributes A_i among the attributes $\{A_1, A_2, \cdots, A_n\}$ from the training data D with highest information gain value, Then split the training data D into sub-datasets $\{D_1, D_2, \cdots, D_n\}$ depending on the chosen attribute values of A_i. After the algorithm estimates the prior and conditional probabilities for each sub-dataset $D_i = \{D_1, D_2, \cdots, D_n\}$ and classifies the examples of each sub-dataset D_i using their respective probabilities. If any example of any sub-dataset D_i is misclassified then the algorithm calculates the information gain of attributes for that sub-dataset D_i, and chooses the best attribute A_i with maximum information gain value from sub-dataset D_i, and split the sub-dataset D_i into sub-sub-datasets D_{ij}. Then again calculates the prior and conditional probabilities for each sub-sub-dataset D_{ij}, and also classifies the examples of sub-sub-datasets using their respective probabilities. The algorithm will continue this process until all the examples of sub/sub-sub-datasets are correctly classified. When the algorithm correctly classifies all the examples then the prior and conditional probabilities for each datasets are preserved for future classification of unseen examples. The main procedure of the algorithm is described in Algorithm 1.

```
Algorithm 1 NBDTAID Algorithm
Input: Training Data, $D$.
Output: Adaptive Intrusion Detection Model, $AIDM$.
Procedure:
 1: Search the multiple copies of same example in $D$, if found then keeps only one unique
    example in $D$.
 2: For each continuous attributes in $D$ find the each adjacent pair of continuous attribute
    values that are not classified into the same class value for that continuous attribute.
 3: Calculate the prior probabilities $P(C_j)$ and conditional probabilities $P(A_{ij}|C_j)$ in $D$.
 4: Classify all the training examples using these prior and conditional probabilities,
        $P(e_i|C_j) = P(C_j) \prod_{k=1}^{p} P(A_{ij}|C_j)$
 5: Update the class value for each example in $D$ with Maximum Likelihood (ML) of posterior
    probability,
        $P(C_j|e_i); C_j = C_i \to P_{ML}(C_j|e_i)$
 6: Recalculate the prior $P(C_j)$ and conditional $P(A_{ij}|C_j)$ probabilities using updated class
    values in $D$.
 7: Again classify all training examples in $D$ using updated probability values.
 8: If any training examples in $D$ is misclassified then calculate the information gain for each
    attributes $A_i = \{A_1, A_2, \cdots, A_n\}$ in $D$ using equation 11.
 9: Choose the best attribute $A_i$ from $D$ with the maximum information gain value.
10: Split dataset $D$ into sub-datasets $\{D_1, D_2, \cdots, D_n\}$ depending on the attribute values of
    $A_i$.
11: Calculate the prior $P(C_j)$ and conditional $P(A_{ij}|C_j)$ probabilities of each sub-dataset $D_i$.
12: Classify the examples of each sub-dataset $D_i$ with their respective prior and conditional
    probabilities.
13: If any example of any sub-dataset $D_i$ is misclassified then calculate the information gain
    of attributes for that sub-dataset $D_i$, and choose one best attribute $A_i$ with maximum gain
    value, then split the sub-dataset $D_i$ into sub-sub-datasets $D_{ij}$. Then again calculate the
    probabilities for each sub-sub-dataset $D_{ij}$. Also classify the examples in sub-sub-datasets
    using their respective probabilities.
14: Continue this process until all the examples are correctly classified.
15: Preserved all the probabilities of each dataset for future classification of examples.
```

3.2. Intrusions Classification using Decision Tree

Attacks Classificaton using Decision Tree (ACDT) for anomaly based network intrusion detection [13] addresses the problem of attacks classification in intrusion detection for classifying different types of network attacks.

In a given dataset, first the ACDT algorithm initializes the weights for each example of dataset; W_i equal to $\frac{1}{n}$, where n is the number of total examples in dataset. Then the ACDT algorithm estimates the prior probability $P(C_j)$ for each class by summing the weights that how often each class occurs in the dataset. Also for each attribute, A_i, the number of occurrences of each attribute value A_{ij} can be counted by summing the weights to determine $P(A_{ij})$. Similarly, the conditional probabilities $P(A_{ij}|C_j)$ are estimated for all values of attributes by summing the weights how often each attribute value occurs in the class C_j. After that the ACDT algorithm uses these probabilities to update the weights for each example in the dataset. It is performed by multiplying the probabilities of the different attribute values from the examples. Suppose the example e_i has independent attribute values $\{A_{i1}, A_{i2}, \cdots, A_{ip}\}$. We already know $P(A_{ik}|C_j)$, for each class C_j and attribute A_{ik}. We then estimate $P(e_i|C_j)$ by

$$P(e_i|C_j) = P(C_j)\prod_{k=1}^{p} P(A_{ij}|C_j)$$

To update the weight, the algorithm estimate the likelihood of e_i in each class C_j. The probability that e_i is in a class is the product of the conditional probabilities for each attribute value. The posterior probability $P(C_j|e_i)$ is then found for each class. Now the weight of the example is updated with the highest posterior probability for that example. Finally, the algorithm calculates the information gain by using updated weights and builds a tree for decision making. Algorithm 2 describes the main procedure of learning process:

Algorithm 2 ACDT Algorithm

```
Input: Dataset, $D$.
Output: Decision Tree, $T$.
Procedure:
1: Initialize all the weights in $D$, $W_i = \frac{1}{n}$, where $n$ is the total number of the examples.
2: Calculate the prior probabilities $P(C_j)$ for each class $C_j$ in $D$.
       $P(C_j) = \frac{\sum_{C_i} W_i}{\sum_{i=1}^{n} W_i}$
3: Calculate the conditional probabilities $P(A_{ij}|C_j)$ for each attribute values in $D$.
       $P(A_{ij}|C_j) = \frac{P(A_{ij})}{\sum_{C_i} W_i}$
4: Calculate the posterior probabilities for each example in $D$.
       $P(e_i|C_j) = P(C_j)\prod_{k=1}^{p} P(A_{ij}|C_j)$
5: Update the weights of examples in $D$ with Maximum Likelihood (ML) of posterior
   probability $P(C_j|e_i)$;
       $W_i = P_{ML}(C_j|e_i)$
6: Find the splitting attribute with highest information gain using the updated weights, $W_i$
   in $D$.
7: $T$ = Create the root node and label with splitting attribute.
8: For each branch of the $T$, $D$ = database created by applying splitting predicate to $D$, and
   continue steps 1 to 7 until each final subset belong to the same class or leaf node created.
9: When the decision tree construction is completed the algorithm terminates.
```

3.3. Attribute Weighting with Adaptive NBTree

This subsection presents learning algorithms: Attribute Weighting Algorithm and Adaptive NBTree Algorithm for reducing FP in intrusion detection [11]. It is based on decision tree based attribute weighting with adaptive naïve Bayesian tree (NBTree), which not only reduce FP at acceptable level, but also scale up DR for different types of network intrusions. It estimate the degree of attribute dependency by constructing decision tree, and considers the depth at which attributes are tested in the tree. In NBTree nodes contain and split as regular decision tree, but the leaves contain naïve Bayesian classifier. The purpose of this subsection is to identify important input attributes for intrusion detection that is computationally efficient and effective.

3.3.1. Attribute Weighting Algorithm

In a given training data, $D = \{A_1, A_2, \cdots, A_n\}$ of attributes, where each attribute $A_i = \{A_{i1}, A_{i2}, \cdots, A_{ik}\}$ contains attribute values and a set of classes $C = \{C_1, C_2, \cdots, C_n\}$, where each class $C_j = \{C_{j1}, C_{j2}, \cdots, C_{jk}\}$ has some values. Each example in the training data contains weight, $w = \{w_1, w_2, \cdots, w_n\}$. Initially, all the weights of examples in training data have equal unit value that set to $w_i = \frac{1}{n}$. Where n is the total number of training examples. Estimates the prior probability $P(C_j)$ for each class by summing the weights that how often each class occurs in the training data. For each attribute, A_i, the number of occurrences of each attribute value A_{ij} can be counted by summing the weights to determine $P(A_{ij})$. Similarly, the conditional probability $P(A_{ij}|C_j)$ can be estimated by summing the weights that how often each attribute value occurs in the class C_j in the training data. The conditional probabilities $P(A_{ij}|C_j)$ are estimated for all values of attributes. The algorithm then uses the prior and conditional probabilities to update the weights. This is done by multiplying the probabilities of the different attribute values from the examples. Suppose the training example e_i has independent attribute values $\{A_{i1}, A_{i2}, \cdots, A_{ip}\}$. We already know the prior probabilities $P(C_j)$ and conditional probabilities $P(A_{ik}|C_j)$, for each class C_j and attribute A_{ik}. We then estimate $P(e_i|C_j)$ by

$$P(e_i|C_j) = P(C_j) \prod P(A_{ij}|C_j) \tag{12}$$

To update the weight of training example e_i, we can estimate the likelihood of e_i for each class. The probability that e_i is in a class is the product of the conditional probabilities for each attribute value. The posterior probability $P(C_j|e_i)$ is then found for each class. Then the weight of the example is updated with the highest posterior probability for that example and also the class value is updated according to the highest posterior probability. Now, the algorithm calculates the information gain by using updated weights and builds a tree. After the tree construction, the algorithm initialized weights for each attributes in training data D. If the attribute in the training data is not tested in the tree then the weight of the attribute is initialized to 0, else calculates the minimum depth, d that the attribute is tested at and initialized the weight of attribute to $\frac{1}{\sqrt{d}}$. Finally, the algorithm removes all the attributes with zero weight from the training data D. The main procedure of the algorithm is described in Algorithm 3.

3.3.2. Adaptive NBTree Algorithm

Given training data, D where each attribute A_i and each example e_i have the weight value. Estimates the prior probability $P(C_j)$ and conditional probability $P(A_{ij}|C_j)$ from the given

Algorithm 3 Attribute Weighting Algorithm

```
Input: Training Dataset, $D$.
Output: Decision tree, $T$.
Procedure:
 1: Initialize all the weights for each example in $D$, $w_i = \frac{1}{n}$, where $n$ is the total number of the
    examples.
 2: Calculate the prior probabilities $P(C_j)$ for each class $C_j$ in $D$.
        $P(C_j) = \frac{\sum_{C_i} W_i}{\sum_{i=1}^{n} W_i}$
 3: Calculate the conditional probabilities $P(A_{ij}|C_j)$ for each attribute values in $D$.
        $P(A_{ij}|C_j) = \frac{P(A_{ij})}{\sum_{C_i} W_i}$
 4: Calculate the posterior probabilities for each example in $D$.
        $P(e_i|C_j) = P(C_j) \prod P(A_{ij}|C_j)$
 5: Update the weights of examples in $D$ with Maximum Likelihood $(ML)$ of posterior
    probability $P(C_j|e_i)$;
        $W_i = P_{ML}(C_j|e_i)$
 6: Change the class value of examples associated with maximum posterior probability,
        $C_j = C_i \rightarrow P_{ML}(C_j|e_i)$
 7: Find the splitting attribute with highest information gain using the updated weights, $W_i$
    in $D$.
        $InformationGain = \left( -\sum_{j=1}^{k} \frac{\sum_{i=C_i} W_i}{\sum_{i=1}^{n} W_i} log \frac{\sum_{i=C_i} W_i}{\sum_{i=1}^{n} W_i} \right) - \left( \sum_{i=1}^{n} \frac{\sum_{i=C_{ij}} W_i}{\sum_{i=C_i} W_i} log \left( \sum_{i=C_{ij}} W_i \right) \right)$
 8: $T$ = Create the root node and label with splitting attribute.
 9: For each branch of the $T$, $D$ = database created by applying splitting predicate to $D$, and
    continue steps 1 to 8 until each final subset belong to the same class or leaf node created.
10: When the decision tree construction is completed, for each attribute in the training data
    $D$: If the attribute is not tested in the tree then weight of the attribute is initialized to 0.
    Else, let $d$ be the minimum depth that the attribute is tested in the tree, and weight of the
    attribute is initialized to $\frac{1}{\sqrt{d}}$.
11: Remove all the attributes with zero weight from the training data $D$.
```

training dataset using weights of the examples. Then classify all the examples in the training dataset using these prior and conditional probabilities with incorporating attribute weights into the naïve Bayesian formula:

$$P(e_i|C_j) = P(C_j)\prod_{i=1}^{m} P(A_{ij}|C_j)^{W_i} \tag{13}$$

Where W_i is the weight of attribute A_i. If any example of training dataset is misclassified, then for each attribute A_i, evaluate the utility, $u(A_i)$, of a spilt on attribute A_i. Let $j = argmax_i(u_i)$, i.e., the attribute with the highest utility. If u_j is not significantly better than the utility of the current node, create a NB classifier for the current node. Partition the training data D according to the test on attribute A_i. If A_i is continuous, a threshold split is used; if A_i is discrete, a multi-way split is made for all possible values. For each child, call the algorithm recursively on the portion of D that matches the test leading to the child. The main procedure of the algorithm is described in Algorithm 4.

Algorithm 4 Adaptive NBTree Algorithm

```
Input: Training dataset $D$ of labeled examples.
Output: A hybrid decision tree with naïve Bayesian classifier at the leaves.
Procedure:
  1: Calculate the prior probabilities $P(C_j)$ for each class $C_j$ in $D$.
         $P(C_j) = \frac{\sum_{C_i} W_i}{\sum_{i=1}^{n} W_i}$
  2: Calculate the conditional probabilities $P(A_{ij}|C_j)$ for each attribute values in $D$.
         $P(A_{ij}|C_j) = \frac{P(A_{ij})}{\sum_{C_i} W_i}$
  3: Classify each example in D with maximum posterior probability.
         $P(e_i|C_j) = P(C_j)\prod_{i=1}^{m} P(A_{ij}|C_j)^{W_i}$
  4: If any example in $D$ is misclassified, then for each attribute $A_i$, evaluate the utility, $u(A_i)$,
     of a spilt on attribute $A_i$.
  5: Let $j = argmax_i(u_i)$, i.e., the attribute with the highest utility.
  6: If $u_j$ is not significantly better than the utility of the current node, create a naÃŕve Bayesian
     classifier for the current node and return.
  7: Partition the training data $D$ according to the test on attribute $A_i$. If $A_i$ is continuous, a
     threshold split is used; if $A_i$ is discrete, a multi-way split is made for all possible values.
  8: For each child, call the algorithm recursively on the portion of $D$ that matches the test
     leading to the child.
```

4. Clustering, boosting, and bagging

In this section, we present IDNBC (Intrusion Detection through naïve Bayesian with Clustering) algorithm [12], Boosting [16], and Bagging [17] algorithms for adaptive intrusion detection. The Boosting algorithm considers a series of classifiers and combines the votes of each individual classifier for classifying intrusions using NB classifier. The Bagging algorithm ensembles ID3, NB classifier, and k-Nearest-Neighbor classifier for intrusion detection, which improves DR and reduces FP. The purpose of this chapter is to combines the several classifiers to improve the classification of different types of network intrusions.

4.1. Naïve Bayesian with clustering

It has been tested that one set of probability derived from data is not good enough to have good classification rate. This subsection presents algorithm namely IDNBC (Intrusion Detection through naïve Bayesian with Clustering) for mining network logs to detect network intrusions through NB classifier [12], which clusters the network logs into several groups based on similarity of logs, and then calculates the probability set for each cluster. For classifying a new log, the algorithm checks in which cluster the log belongs and then use that cluster probability set to classify the new log.

Given a database $D = \{t_1, t_2, \cdots, t_n\}$ where $ti = \{t_{i1}, t_{i2}, \cdots, t_{ih}\}$ and the database D contains the following attributes $\{A_1, A_2, \cdots, A_n\}$ and each attribute A_i contains the following attribute values $\{A_{i1}, A_{i2}, \cdots, A_{ih}\}$. The attribute values can be discrete or continuous. Also the database D contains a set of classes $C = \{C_1, C_2, \cdots, C_m\}$. Each example in the database D has a particular class C_j. The algorithm first clusters the database D into several clusters $\{D_1, D_2, \cdots, D_n\}$ depending on the similarity of examples in the database D. A similarity measure, $sim(t_i, t_l)$, defined between any two examples, t_1, t_2 in D, and an integer value k, the clustering is to define a mapping $f : D \rightarrow \{1, \cdots, K\}$ where each t_i is assigned to one cluster K_j. Suppose for two examples there is a match between two attribute values then the similarity becomes 0.5. If there is a match only in one attribute value, then similarity between the examples is taken as 0.25 and so on. Then the algorithm calculates the prior probabilities $P(C_j)$ and conditional probabilities $P(A_{ij}|C_j)$ for each cluster. The prior probability $P(C_j)$ for each class is estimated by counting how often each class occurs in the cluster. For each attribute A_i the number of occurrences of each attribute value A_{ij} can be counted to determine $P(A_i)$. Similarly, the conditional probability $P(A_{ij}|C_j)$ for each attribute values A_{ij} can be estimated by counting how often each attribute value occurs in the class in the cluster. For classifying a new example whose attribute values are known but class value is unknown, the algorithm checks in which cluster the new example belongs and then use that cluster probability set to classify the new example. For classifying a new example, the prior probabilities and conditional probabilities are used to make the prediction. This is done by combining the effects of the different attribute values from that example. Suppose the example e_i has independent attribute values $\{A_{i1}, A_{i2}, \cdots, A_{ip}\}$, we know the $P(A_{ik}|C_j)$, for each class C_j and attribute A_{ik}. We then estimate $P(e_i|C_j)$ to classify the example, the probability that e_i is in a class is the product of the conditional probabilities for each attribute value with prior probability for that class. The posterior probability $P(C_j|e_i)$ is then found for each class and the example classifies with the highest posterior probability for that example. The main procedure of the algorithm is described in Algorithm 5.

Algorithm 5 IDNBC Algorithm

```
Input: Database, $D$.
Output: Intrusion Detection Model.
Procedure:
1: for each example $t_i \epsilon D$, check the similarity of examples: $sim(t_i, t_l)$;
2: Put examples into cluster: $D_i \leftarrow t_i$;
3: for each cluster $D_i$, calculate the prior probabilities:
       $$P(C_j) = \frac{\sum t_{i \to C_j}}{\sum_{i=1}^{n} t_i}$$
4: for each cluster $D_i$, calculate the conditional probabilities:
       $$P(A_{ij}|C_j) = \frac{\sum_{i=1}^{n} A_{i \to C_j}}{\sum t_{i \to C_j}}$$
5: for each cluster $D_i$, store the prior probabilities, $S_1 = P(C_j)$; and conditional probabilities,
   $S_2 = P(A_{ij}|C_j)$;
6: For classifying new example, check in which cluster the example belongs and then use
   that cluster probability set to classify the example.
```

4.2. Boosting

Adaptive intrusion detection using boosting and naïve Bayesian classifier [16], which considers a series of classifiers and combines the votes of each individual classifier for classifying an unknown or known intrusion. This algorithm generates the probability set for each round using naïve Bayesian classifier and updates the weights of training examples based on the misclassification error rate that produced by the training examples in each round.

Given a training data $D = \{t_1, \cdots, t_n\}$, where $t_i = \{t_{i1}, \cdots, t_{ih}\}$ and the attributes $\{A_1, A_2, \cdots, A_n\}$. Each attribute A_i contains the following attribute values $\{A_{i1}, A_{i2}, \cdots, A_{ih}\}$. The training data D also contains a set of classes $C = \{C_1, C_2, \cdots, C_m\}$. Each training example has a particular class C_j. The algorithm first initializes the weights of training examples to an equal value of $w_i = \frac{1}{n}$, where n is the total number of training examples in D. Then the algorithm generates a new dataset D_i with equal number of examples from training data D using selection with replacement technique and calculates the prior $P(C_j)$ and class conditional $P(A_{ij}|C_j)$ probabilities for new dataset D_i.

The prior probability $P(C_j)$ for each class is estimated by counting how often each class occurs in the dataset D_i. For each attribute A_i the number of occurrences of each attribute value A_{ij} can be counted to determine $P(A_i)$. Similarly, the class conditional probability $P(A_{ij}|C_j)$ for each attribute values A_{ij} can be estimated by counting how often each attribute value occurs in the class in the dataset D_i. Then the algorithm classifies all the training examples in training data D with these prior $P(C_j)$ and class conditional $P(A_{ij}|C_j)$ probabilities from dataset D_i. For classifying the examples, the prior and conditional probabilities are used to make the prediction. This is done by combining the effects of the different attribute values from that example. Suppose the example e_i has independent attribute values $\{A_{i1}, A_{i2}, \cdots, A_{ip}\}$, we know $P(A_{ik}|C_j)$, for each class C_j and attribute A_{ik}. We then estimate $P(e_i|C_j)$ by using equation 14.

$$P(e_i|C_j) = P(C_j)\prod_{k=1}^{p} P(A_{ij}|C_j) \tag{14}$$

To classify the example, the probability that e_i is in a class is the product of the conditional probabilities for each attribute value with prior probability for that class. The posterior probability $P(C_j|e_i)$ is then found for each class and the example classifies with the highest posterior probability value for that example. The algorithm classifies each example t_i in D with maximum posterior probability. After that the weights of the training examples ti in training data D are adjusted or updated according to how they were classified. If an example was misclassified then its weight is increased, or if an example was correctly classified then its weight is decreased.

To updates the weights of training data D, the algorithm computes the misclassification rate, the sum of the weights of each of the training example t_i in D that were misclassified. That is,

$$error(M_i) = \sum_{i}^{d} W_i * err(t_i) \tag{15}$$

Where $err(t_i)$ is the misclassification error of example t_i. If the example t_i was misclassified, then is $err(t_i)$ 1. Otherwise, it is 0. The misclassification rate affects how the weights of the training examples are updated. If a training example was correctly classified, its weight is multiplied by error $(\frac{M_i}{1-error(M_i)})$. Once the weights of all of the correctly classified examples are updated, the weights for all examples including the misclassified examples are normalized so that their sum remains the same as it was before. To normalize a weight, the algorithm multiplies the weight by the sum of the old weights, divided by the sum of the new weights. As a result, the weights of misclassified examples are increased and the weights of correctly classified examples are decreased. Now the algorithm generates another new data set D_i from training data D with maximum weight values and continues the process until all the training examples are correctly classified. Or, we can set the number of rounds that the algorithm will iterate the process. To classify a new or unseen example use all the probabilities of each round (each round is considered as a classifier) and consider the class of new example with highest classifier's vote. The main procedure of the boosting algorithm is described in Algorithm 6.

4.3. Bagging

Classification of streaming data based on bootstrap aggregation (bagging) [17] creates an ensemble model by using ID3 classifier, naïve Bayesian classifier, and k-Nearest-Neighbor classifier for a learning scheme where each classifier gives the weighted prediction.

Given a dataset D, of d examples and the dataset D contains the following attributes $\{A_1, A_2, \cdots, A_n\}$ and each attribute A_i contains the following attribute values $\{A_{i1}, A_{i2}, \cdots, A_{ih}\}$. Also the dataset D contains a set of classes $C = \{C_1, C_2, \cdots, C_m\}$, where each example in dataset D has a particular class C_j. The algorithm first generates the training dataset D_i from the given dataset D using selection with replacement technique. It is very likely that some of the examples from the dataset D will occur more than once in the training dataset D_i. The examples that did not make it into the training dataset end up forming the test dataset. Then a classifier model, M_i, is learned for each training examples d from

Algorithm 6 Boosting Algorithm

```
Input: $D$, Training data $D$ of labeled examples $t_i$.
Output: A classification model.
Procedure:
1: Initialize the weight $w_i = \frac{1}{n}$ of each example $t_i$ in $D$, where $n$ is the total number of training examples.
2: Generate a new dataset $D_i$ with equal number of examples from $D$ using selection with replacement technique.
3: Calculate the prior probability $P(C_j)$ for each class $C_j$ in dataset $D_i$:
       $$P(C_i) = \frac{\sum t_{i \to C_j}}{\sum_{i=1}^{n} t_i}$$
4: Calculate the class conditional probabilities $P(A_{ij}|C_j)$ for each attribute values in dataset $D_i$:
       $$P(A_{ij}|C_j) = \frac{\sum_{i=1}^{n} A_{i \to C_j}}{\sum t_{i \to C_j}}$$
5: Classify each training example $t_i$ in training data $D$ with maximum posterior probabilities.
       $$P(e_i|C_j) = P(C_j) \prod_{k=1}^{p} P(A_{ij}|C_j)$$
6: Updates the weights of each training examples $t_i D$, according to how they were classified. If an example was misclassified then its weight is increased, or if an example was correctly classified then its weight is decreased. To updates the weights of training examples the misclassification rate is calculated, the sum of the weights of each of the training example $t_i D$ that were misclassified: $error(M_i) = \sum_i^d W_i * err(t_i)$; Where $err(t_i)$ is the misclassification error of example $t_i$. If the example $t_i$ was misclassified, then is $err(t_i)$ 1. Otherwise, it is 0. If a training example was correctly classified, its weight is multiplied by $(\frac{M_i}{1 - error(M_i)})$. Once the weights of all of the correctly classified examples are updated, the weights for all examples including the misclassified examples are normalized so that their sum remains the same as it was before. To normalize a weight, the algorithm multiplies the weight by the sum of the old weights, divided by the sum of the new weights. As a result, the weights of misclassified examples are increased and the weights of correctly classified examples are decreased.
7: Repeat steps 2 to 6 until all the training examples $t_i$ in $D$ are correctly classified.
8: To classify a new or unseen example use all the probability set in each round (each round is considered as a classifier) and considers the class of new example with highest classifier's vote.
```

training dataset D_i. The algorithm builds three classifiers using ID3, naïve Bayesian (NB), and k-Nearest-Neighbor (kNN) classifiers.

The basic strategy used by ID3 classifier is to choose splitting attributes with the highest information gain first and then builds a decision tree. The amount of information associated with an attribute value is related to the probability of occurrence. The concept used to quantify information is called entropy, which is used to measure the amount of randomness from a data set. When all data in a set belong to a single class, there is no uncertainty, and then the entropy is zero. The objective of decision tree classification is to iteratively partition the given data set into subsets where all elements in each final subset belong to the same class. The entropy calculation is shown in equation 16. Given probabilities $p_1, p_2, \cdots, p_s$ for different classes in the data set

$$Entropy : H(p_1, p_2, \cdots, p_s) = \sum_{i=1}^{s}(p_i log(\frac{1}{p_i})) \tag{16}$$

Given a data set, D, $H(D)$ finds the amount of entropy in class based subsets of the data set. When that subset is split into s new subsets $S = D_1, D_2, \cdots, D_s$ using some attribute, we can again look at the entropy of those subsets. A subset of data set is completely ordered and does not need any further split if all examples in it belong to the same class. The ID3 algorithm calculates the information gain of a split by using equation 17 and chooses that split which provides maximum information gain.

$$Gain(D, S) = H(D) - \sum_{i=1}^{s} p(D_i)H(D_i) \tag{17}$$

The naïve Bayesian (NB) classifier calculates the prior probability, $P(C_j)$ and class conditional probability, $P(A_{ij}|C_j)$ from the dataset. For classifying an example, the NB classifier uses these prior and conditional probabilities to make the prediction of class for that example. The prior probability $P(C_j)$ for each class is estimated by counting how often each class occurs in the dataset D_i. For each attribute A_i the number of occurrences of each attribute value A_{ij} can be counted to determine $P(A_i)$. Similarly, the class conditional probability $P(A_{ij}|C_j)$ for each attribute values A_{ij} can be estimated by counting how often each attribute value occurs in the class in the dataset D_i.

The k-Nearest-Neighbor (kNN) classifier assumes that the entire training set includes not only the data in the set but also the desired classification for each item. When a classification is to be made for a test or new example, its distance to each item in the training data must be determined. The test or new example is then placed in the class that contains the most examples from this training data of k closest items.

After building classifiers using ID3, NB, and kNN, each classifier, M_i, classifies the training examples and initialized the weight, W_i of each classifier based on the accuracies of percentage of correctly classified examples from training dataset. To classify the testing examples or unknown examples each classifier returns its class prediction, which counts as one vote. The proposed bagged classifier counts the votes with the weights of classifiers, and assigns the class with the maximum weighted vote. The main procedure of the bagging algorithm is described in Algorithm 7.

Algorithm 7 Bagging Algorithm

```
Input:
D, a set of d examples.
k = 3, the number of models in the ensemble.
Learning scheme (ID3, naïve Bayesian classifier, and k-Nearest-Neighbor).
Output: A composite model, $M*$.
Procedure:
1: Generate a new training dataset $D_i$ with equal number of examples from a given dataset
   $D$ using selection with replacement technique. Same example from given dataset $D$ may
   occur more than once in the training dataset $D_i$.
2: for i = 1 to k do
3: Derive a model or classifier, $M_i$ using training dataset $D_i$.
4: Classify each example $d$ in training data $D_i$ and initialized the weight, $W_i$ for the model,
   $M_i$, based on the accuracies of percentage of correctly classified example in training data
   $D_i$.
5: endfor
To use the composite model on test examples or unseen examples:

1: for i = 1 to k do
2: Classify the test or unseen examples using the $k$ models.
3: Returns a weighted vote (which counts as one vote).
4: endfor
5: $M*$, counts the votes and assigns the class with the maximum weighted vote for that
   example.
```

5. Experimental results

The experiments were performed by using an Intel Core 2 Duo Processor 2.0 GHz processor (2 MB Cache, 800 MHz FSB) with 1 GB of RAM.

5.1. NBDTAID evaluation

In order to evaluate the performance of NBDTAID algorithm for network intrusion detection, we performed 5-class classification using KDD99 intrusion detection benchmark dataset [14]. The results of the comparison of NBDTAID with naïve Bayesian classifier and ID3 classifier are presented in Table 5 using 41 input attributes, and Table 6 using 19 input attributes. The performance of NBDTAID algorithm using reduced dataset (12 and 17 input attributes) increases DR that are summarized in Table 7.

Method	Normal	Probe	DOS	U2R	R2L
NBDTAID (DR %)	99.72	99.25	99.75	99.20	99.26
NBDTAID (FP %)	0.06	0.39	0.04	0.11	6.81
naïve Bayesian (DR %)	99.27	99.11	99.69	64.00	99.11
naïve Bayesian (FP %)	0.08	0.45	0.04	0.14	8.02
ID3 (DR %)	99.63	97.85	99.51	49.21	92.75
ID3 (FP %)	0.10	0.55	0.04	0.14	10.03

Table 5. NBDTAID Algorithm: Comparison of the results using 41 attributes.

Method	Normal	Probe	DOS	U2R	R2L
NBDTAID (DR %)	99.84	99.75	99.76	99.47	99.35
NBDTAID (FP %)	0.05	0.28	0.03	0.10	6.22
naïve Bayesian (DR %)	99.65	99.35	99.71	64.84	99.15
naïve Bayesian (FP %)	0.05	0.32	0.04	0.12	6.87
ID3 (DR %)	99.71	98.22	99.63	86.11	97.79
ID3 (FP %)	0.06	0.51	0.04	0.12	7.34

Table 6. NBDTAID Algorithm: Comparison of the results using 19 attributes.

Class Value	12 Attributes	17 Attributes
Normal	99.98	99.95
Probe	99.92	99.93
DoS	99.99	99.97
U2R	99,38	99.46
R2L	99.55	99.69

Table 7. Performance of NBDTAID algorithm using reduced dataset.

5.2. ACDT evaluation

The results of the comparison of ACDT, ID3, and C4.5 algorithms using 41 attributes are tabulated in Table 8 and using 19 attributes are tabulated Table 9 [13].

Method	Normal	Probe	DoS	U2R	R2L
ACDT (DR %)	98.76	98.21	98.55	98.11	97.16
ACDT (FP %)	0.07	0.44	0.05	0.12	6.85
ID3 (DR %)	97.63	96.35	97.41	43.21	92.75
ID3 (FP %)	0.10	0.55	0.04	0.14	10.03
C4.5 (DR %)	98.53	97.85	97.51	49.21	94.65
C4.5 (FP %)	0.10	0.55	0.07	0.14	11.03

Table 8. Comparison of ACDT with ID3 and C4.5 using 41 Attributes.

Method	Normal	Probe	DoS	U2R	R2L
ACDT (DR %)	99.19	99.15	99.26	98.43	98.05
ACDT (FP %)	0.06	0.48	0.04	0.10	6.32
ID3 (DR %)	98.71	98.22	97.63	86.11	94.19
ID3 (FP %)	0.06	0.51	0.04	0.12	7.34
C4.5 (DR %)	98.81	98.22	97.73	56.11	95.79
C4.5 (FP %)	0.08	0.51	0.05	0.12	8.34

Table 9. Comparison of ACDT with ID3 and C4.5 using 19 Attributes.

5.3. Adaptive NBTree evaluation

Firstly, we used attribute weighting algorithm to perform attribute selection from training dataset of KDD99 dataset and then we used adaptive NBTree algorithm for classifier construction [11]. The performance of our proposed algorithm on 12 attributes in KDD99 dataset is listed in Table 10.

Classes	Detection Rates (%)	False Positives (%)
Normal	100	0.04
Probe	99.93	0.37
DoS	100	0.03
U2R	99,38	0.11
R2L	99.53	6.75

Table 10. Performance of adaptive NBTree algorithm on KDD99 Dataset.

Table 11 and Table 12 depict the performance of naïve Bayesian (NB) classifier and C4.5 algorithm using the original 41 attributes of KDD99 dataset. Table 13 and Table 14 depict the performance of NB classifier and C4.5 using reduces 12 attributes.

Classes	Detection Rates (%)	False Positives (%)
Normal	99.27	0.08
Probe	99.11	0.45
DoS	99.68	0.05
U2R	64.00	0.14
R2L	99.11	8.12

Table 11. Performance of NB classifier on KDD99 Dataset.

Classes	Detection Rates (%)	False Positives (%)
Normal	98.73	0.10
Probe	97.85	0.55
DoS	97.51	0.07
U2R	49.21	0.14
R2L	91.65	11.03

Table 12. Performance of C4.5 algorithm on KDD99 Dataset.

Classes	Detection Rates (%)	False Positives (%)
Normal	99.65	0.06
Probe	99.35	0.49
DoS	99.71	0.04
U2R	64.84	0.12
R2L	99.15	7.85

Table 13. Performance of NB classifier using 12 attributes.

Classes	Detection Rates (%)	False Positives (%)
Normal	98.81	0.08
Probe	98.22	0.51
DoS	97.63	0.05
U2R	56.11	0.12
R2L	91.79	8.34

Table 14. Performance of C4.5 algorithm using 12 attributes.

We compare the detection rates among Support Vector Machines (SVM), Neural Network (NN), Genetic Algorithm (GA), and adaptive NBTree algorithm on KDD99 dataset that tabulated in Table 15.

	SVM	NN	GA	Adaptive NBTree
Normal	99.4	99.6	99.3	99.93
Probe	89.2	92.7	98.46	99.84
DoS	94.7	97.5	99.57	99.91
U2R	71.4	48	99.22	99.47
R2L	87.2	98	98.54	99.63

Table 15. Comparison of several algorithms with adaptive NBTree algorithm.

5.4. IDNBC evaluation

The performance of IDNBC algorithm tested by employing KDD99 benchmark network intrusion detection dataset, and the experimental results proved that it improves DR as well as reduces FP for different types of network intrusions are tabulated in Table 16 [12].

Method	Normal	Probe	DoS	U2R	R2L
IDNBC (DR %)	99.66	99.24	99.62	99.19	99.08
IDNBC (FP %)	0.08	0.86	0.09	0.18	7.85
NB (DR %)	99.27	99.11	99.69	64.00	99.11
NB (FP %)	0.08	0.45	0.05	0.14	8.02

Table 16. Performance of IDNBC algorithm with naïve Bayesian classifier.

5.5. Boosting evaluation

We tested the performance of Boosting algorithm with k-Nearest-Neighbor classifier (kNN), Decision Tree classifier (C4.5), Support Vector Machines (SVM), Neural Network (NN), and Genetic Algorithm (GA) by employing on the KDD99 benchmark intrusion detection dataset [16] that is tabulated in Table 17.

Method	Normal	Probe	DoS	U2R	R2L
Boosting Algorithm	100	99.95	99.92	99.55	99.60
kNN	99.60	75.00	97.30	35.00	0.60
C4.5	98.49	94.82	97.51	49.25	91.26
SVM	99.40	89.2	94.7	71.40	87.20
NN	99.60	92.7	97.50	48.00	98.00
GA	99.30	98.46	99.57	99.22	98.54

Table 17. Comparison of the results for the intrusion detection problem (Detection Rate %).

It has been successfully tested that effective attributes selection improves the detection rates for different types of network intrusions in intrusion detection. The performance of boosting algorithm on 12 attributes in KDD99 dataset is listed in Table 18.

Attack Types	DR (%)	FP (%)
Normal	100	0.03
Probing	99.95	0.36
DoS	100	0.03
U2R	99.67	0.10
R2L	99.58	6.71

Table 18. Boosting on reduce KDD99 dataset.

5.6. Bagging evaluation

The presented bagging algorithm was tested on the KDD99 benchmark intrusion detection dataset that is tabulated in Table 19 [17].

Method	Normal	Probe	DoS	U2R	R2L
ID3	99.63	97.85	99.51	49.21	92.75
NB	99.27	99.11	99.69	64.00	99.11
kNN	99.60	75.00	97.30	35.00	0.60
Bagging Algorithm	100	99.92	99.93	99.57	99.61

Table 19. Comparison of the results on KDD99 dataset using bagging (Detection Rate %).

6. Conclusions and future work

The work presented in this chapter has explored the basic concepts of adaptive intrusion detection employing data mining algorithms. We focused on naïve Bayesian (NB) classifier and decision tree (DT) classifier for extracting intrusion patterns from network data. Both NB and DT are efficient learning techniques for mining the complex data and already applied in many real world problem domains. NB has several advantages. First, it is easy to use. Second, unlike other learning algorithms, only one scan of the training data is required. NB can easily handle missing attribute values by simply omitting that probability when calculation the likelihoods of membership in each class. On the other side, the ID3 algorithm to build a decision tree based on information theory and attempts to minimize the expected number of comparisons. The basic strategy used by ID3 is to choose splitting attributes with the highest information gain. The amount of information associated with an attribute value is related to the probability of occurrence. Having evaluated the mining algorithms on KDD99 benchmark intrusion detection dataset, it proved that supervised intrusion classification can increased DR and significantly reduced FP. It also proved that data mining for intrusion detection works, and the combination of NB classifier and DT algorithm forms a robust intrusion-processing framework. Algorithms such as NBDTAID, ACDT, Attribute Weighting with Adaptive NBTree, IDNBC, Boosting, and Bagging presented in this chapter can increase the DR and significantly reduce the FP in intrusion detection. The future works focus on improving FP for R2L attacks and ensemble with other mining algorithms to improve the DR for new network attacks.

Acknowledgements

Support for this research received from the National Science & Information and Communication Technology (NSICT), Ministry of Science and Information & Communication Technology, Government of Bangladesh, Department of Computer Science and Engineering, Jahangirnagar University, Bangladesh, and Department of Computer Science and Engineering, United International University, Bangladesh.

Author details

Dewan Md. Farid and Mohammad Zahidur Rahman
Department of Computer Science and Engineering, Jahangirnagar University, Bangladesh

Chowdhury Mofizur Rahman
Department of Computer Science and Engineering, United International University, Bangladesh

7. References

[1] Anderson, J. P. [1980]. Computer security threat monitoring and surveillance, *Technical report 98-17, James P. Anderson Co., Fort Washington, Pennsylvania, USA* .

[2] Aydin, M. A., Zaim, A. H. & Ceylan, K. G. [2009]. A hybrid intursion detection system design for computer network security, *Computers & Electrical Engineering* 35(3): 517–526.

[3] Bankovic, Z., Stepanovic, D., Bojanic, S. & Talasriz, O. N. [2007]. Improving network security using genetic algorithm approach, *Computers & Electrical Engineering* 33(5-6): 438–541.

[4] BarbarÃą, D., Couto, J., Jajodia, S. & Wu, N. [2001]. ADAM: A tested for exploring the use of data mining in intrusion detection, *Special Interest Group on Management of Data (SIGMOD)* 30(4): 15–24.

[5] Barbara, D., Couto, J., Jajodia, S., Popyack, L. & Wu, N. [2001]. ADAM: Detecting intrusion by data mining, *In Proc. of the 2001 IEEE Workshop on Information Assurance and Security, United States Military Academy, West Point* pp. 11–16.

[6] Bass, T. [2000]. Intrusion detection systems and multi-sensor data fusion, *Communications of the ACM* 43(4): 99–105.

[7] Chebrolu, S., Abraham, A. & Thomas, J. P. [2004]. Feature deduction and ensemble design of intrusion detection systems, *Computer & Security* 24(4): 295–307.

[8] Chen, R. C. & Chen, S. P. [2008]. Intrusion detection using a hybrid support vector machine based on entropy and TF-IDF, *International Journal of Innovative Computing, Information, and Control (IJICIC)* 4(2): 413–424.

[9] Chen, W. H., Hsu, S. H. & Shen, H. P. [2005]. Application of SVM and ANN for intrusion detection, *Computers & Operations Research* 32(10): 2617–1634.

[10] Depren, O., Topallar, M., Anarim, E. & Ciliz, M. K. [2006]. An intelligence intrusion detection system for anomaly and misuse detection in computer networks, *Expert Systems with Applications* 29: 713–722.

[11] Farid, D. M., Darmont, J. & Rahman, M. Z. [2010]. Attribute weighting with adaptive nbtree for reducing false positives in intrusion detection, *International Journal of Computer Science and Information Security (IJCSIS)* 8(1): 19–26.

[12] Farid, D. M., Harbi, N., Ahmmed, S., Rahman, M. Z. & Rahman, C. M. [2010]. Mining network data for intrusion detection through naïve bayesian with clustering, *In Proc. of the International Conference on Computer, Electrical, System Science, and Engineering (ICCESSE 2010), Paris, France* pp. 836–840.

[13] Farid, D. M., Harbi, N., Bahri, E., Rahman, M. Z. & Rahman, C. M. [2010]. Attacks classification in adaptive intrusion detection using decision tree, *In Proc. of the International Conference on Computer Science (ICCS 2010), Rio De Janeiro, Brazil* pp. 86–90.

[14] Farid, D. M., Harbi, N. & Rahman, M. Z. [2010]. Combining naïve bayes and decision tree for adaptive intrusion detection, *International Journal of Network Security and Its Applications (IJNSA)* 2(2): 12–25.

[15] Farid, D. M., Hoa, N. H., Darmont, J., Harbi, N. & Rahman, M. Z. [2010]. Scaling up detection rates and reducing false positives in intrusion detection using nbtree, *In Proc. of the International Conference on Data Mining and Knowledge Engineering (ICDMKE 2010), Rome, Italy* pp. 186–190.

[16] Farid, D. M., Rahman, M.-Z. & Rahman, C. M. [2011a]. Adaptive intrusion detection based on boosting and naïve bayesian classifier, *International Journal of Computer Applications (IJCA)* 24(3): 12–19. Published by Foundation of Computer Science, New York, USA.

[17] Farid, D. M., Rahman, M. Z. & Rahman, C. M. [2011b]. An ensemble approach to classifier construction based on bootstrap aggregation, *International Journal of Computer Applications (IJCA)* 25(5): 30–34. Published by Foundation of Computer Science, New York, USA.

[18] Forn, C. & Lin, C. Y. [2010]. A triangle area based nearset neighbors approach to intrusion detection, *Pattern Recognition* 43(1): 222–229.

[19] Foster, J. J. C., Jonkman, M., Marty, R. & Seagren, E. [2006]. Intrusion detection systems, *Snort Intrusion detection and Prevention Toolkit* pp. 1–30.

[20] Heberlein & Let, A. [1990]. A network secutiry mnitor, *In Proc. of the IEEE Computer Society Symposium, Research in Security and Privacy* pp. 296–303.

[21] Helmer, G., Wong, J. S. K., Honavar, V. & Miller, L. [2002]. Automated discovery of concise predictive rules for intrusion detection, *Journal of Systems and Software* 60(3): 165–175.

[22] Jinquan, Z., Xiaojie, L., Tao, L., Caiming, L., Lingxi, P. & Feixian, S. [2009]. A self-adaptive negative selection algorithm used for anomaly detection, *Progress in Natural Science* 19(2): 261–266.

[23] Patch, A. & Park, J. M. [2007]. An overview of anomaly detection techniques: Existing solutions and latest technological trends, *Computer Netwroks* 51(12): 3448–3470.

[24] Paxson, V. [1999]. Bro: A system for detecting network intruders in real-time, *Computer Networks* pp. 2435–2463.

[25] Provost, F. & Fawcett, T. [2001]. Robust classification for imprecise environment, *Machine Learning* 42(3): 203–231.

[26] Rexworthy, B. [2009]. Intrusion detections systems âĂŞ an outmoded network protection model, *Network Security* 2009(6): 17–19.

[27] Seredynski, F. & Bouvry, P. [2007]. Anomaly detection in TCP/IP networks using immune systems paradigm, *Computer Communications* 30(4): 740–749.

[28] Teodoro, P. G., Verdijo, J. D., Fernandez, G. M. & Vazquez, E. [2009]. Anomaly-based network intrusion detection: Techniques, systems and challenges, *Computers & Security* 28(1-2): 18–28.

[29] *The KDD Archive. KDD99 cup dataset* [1999].
http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.

[30] Tsai, C. F., Hsu, Y. F., Lin, C. Y. & Lin, W. Y. [2009]. Intrusion detection by machine learning: A review, *Expert Systems with Applications* 36(10): 11994–12000.

[31] Tsymbal, A., Puuronen, S. & Patterson, D. W. [2003]. Ensemble feature selection with the simple bayesian classification, *Information Fusion* 4(2): 87–100.

[32] Wang, W., Guan, X. & Zhang, X. [2008]. Processing of massive audit data streams for real-time anomaly intrusion detection, *Computer Communications* 31(1): 58–72.

[33] Wu, H. C. & Huand, S. H. S. [2010]. Neural network-based detection of stepping-stone intrusion, *Expert Systems with Applications* 37(2): 1431–1437.

[34] Wuu, L. C., Hung, C. H. & Chen, S. F. [2007]. Building intrusion pattern miner for Snort network intrusion detection system, *Journal of Systems and Software* 80(10): 1699–1715.

[35] Yeung, D. Y. & Ding, Y. X. [2003]. Host-based intrusion detection using dynamic and static behavioral models, *Pattern Recognition* 36: 229–243.