Chapter 5
DS-AdaBoost Algorithm

5.1 Introduction
The basic aim of this work is to construct an efficient algorithm for pattern based intrusion detection which can achieve high detection rate and low false alarm rate. The motivation for applying the AdaBoost algorithm includes: 1) The AdaBoost algorithm is one of the most popular machine learning algorithms. Its theoretical basis is sound, and its implementation is simple. It has been applied to many pattern recognition problems, such as face recognition. However, the application of the AdaBoost algorithm to intrusion detection has not been explored so far. 2) The AdaBoost algorithm corrects the misclassifications made by weak classifiers, and it is less susceptible to over-fitting than most learning algorithms. Pattern recognition performances of the AdaBoost-based classifiers are generally encouraging. 3) Data sets for intrusion detection are a heterogeneous mixture of categorical and continuous types of features. The different feature types in such data sets make it difficult to find relations between these features. By combining the weak classifiers for continuous features and the weak classifiers for categorical features into a strong classifier, the relations between these two different types of features are handled naturally, without any forced conversions between continuous and categorical features. 4) If simple weak classifiers are used, the AdaBoost algorithm is very fast.

5.2 AdaBoost Algorithm
Machine learning studies automatic techniques for learning to make accurate predictions based on past observations. The goal, of course, is to generate a rule that makes the most accurate predictions possible on new test examples.
Building a highly accurate prediction rule for various patterns is certainly a difficult task. On the other hand, it is not hard at all to come up with very rough rules of thumb that are only moderately accurate. An example of such a rule is something like the following: "If the phrase 'buy now' occurs in the email, then predict it is spam." Such a rule will not even come close to covering all spam messages; for instance, it really says nothing about what to predict if 'buy now' does not occur in the message. On the other hand, this rule will make predictions that are significantly better than random guessing.

Boosting is an iterative process, which adaptively changes the distribution of training examples so that the base classifiers will focus on examples that are hard to classify. It has its roots in a theoretical framework for studying machine learning called the "PAC" learning model [149, 150]. The concept of adaptive boosting called AdaBoost algorithm was first introduced by Freund and Schapire in 1997 that classify an example by voting the weighted predictions of a set of base classifiers, which are generated in a series of rounds [151]. The main idea of boosting algorithms is combining many simple and moderately accurate hypotheses called weak classifiers into a single, highly accurate classifier called strong classifier for the task at hand.

Boosting have become one of the alternative framework for classifier design, together with the more established classifiers, like Bayesian classifier, decision tree, neural network, and support vector machine. Boosting assigns a weight to each training example and adaptively changes the weight at the end of each boosting round. A sample is drawn according to the sampling distribution of the training examples to obtain a new training dataset. Next, a classifier is induced from the training dataset and used to classify all the examples in the original dataset. The weights of the training examples are updated at the end of each boosting round. Examples that are misclassified will have their weights increased, while those that are correctly classified will have their weight decreased. This forces the classifier to focus on examples that are difficult to classify in subsequent iterations.

To combine the weak classifiers iteratively, the AdaBoost algorithm which is one of the best learning algorithms is used. It has inspired several learning theoretical results. Due to its simplicity, flexibility, and excellent performance on real-world data, AdaBoost has gained popularity among researchers [151].
5.3 Architecture

One approach to design Intrusion Detection System is to define network patterns behaviours that indicate intrusion and look for the occurrence of those patterns. While such an approach may be capable of detecting known varieties of intrusions or attacks, it will allow new or unseen types of attacks to go undetected. As a result our decision is to build a system which learn from the normal patterns and detect the intrusion or attacks from the learned patterns.

According to the characteristics of the intrusion detection problem, the research work is split into four phases to facilitate through: analysis, design, validation and evaluation for the selection of best model for IDS, as shown in Figure 5.1.

- Phase - 1 : Feature Extraction and Preprocessing
- Phase - 2 : Data Labeling
- Phase - 3 : Design of the weak classifier
- Phase - 4 : Construction of the strong classifier

Figure 5.1: System Architecture
5.3.1 Feature Extraction
A feature extraction module is used to convert the raw data into feature vectors that can be processed by machine learning algorithms. Feature extraction is the basis for high-performance intrusion detection using machine learning methods since the detection models have to be optimized based on the selection of feature spaces. If the features are improperly selected, the ultimate performance of detection models will be influenced a lot. This problem has been studied during the early work of W.K. Lee [152] and his research results lead to the benchmark dataset of KDD99, where a 41-dimensional feature vector was constructed for each network connection. The feature extraction method in KDD99 made use of various data mining techniques to identify some of the important features for detecting anomalous connections.
In KDD99, there are 494,021 records in the 10% training data set and the number of records in the testing data set is about five million, with a 10 percent testing subset of 311028 records. The data set contains a total amount of 22 different attack types. There are 41 features for each connection record that have either discrete values or continuous values. The 41-dimensional feature can be divided into three groups. The first group of features is called basic or intrinsic features of a network connection, which include the duration, prototype, service, number of bytes from source IP addresses or from destination IP addresses, and some flags in TCP connections. The second group of features in KDD99 is composed of the content features of network connections and the third group is composed of the statistical features that are computed either by a time window or a window of certain kind of connections.

5.3.2 Data labelling
Because the AdaBoost algorithm uses supervised learning, a set of data has to be labelled for training. The normal data is labelled as “+1” and the attack samples are labelled as “-1”.

5.3.3 Design of Weak Classifiers
The AdaBoost algorithm requires a group of weak classifiers designed beforehand. An individual weak classifier is simple and easy to implement. Its classification accuracy is
relatively low. In this the decision stumps for categorical features and continues features are computed separately. Detail design is discussed in section 5.4.

5.3.4 Construction of the Strong Classifiers
A strong classifier is obtained by combining the weak classifiers. The strong classifier has higher classification accuracy than each weak classifier. A strong classifier is trained using the sample labelled data. Then, a new network connection, which is represented by the three groups of features of the network data, is input to the strong classifier and is classified as either “normal” or “attack” as the detection result.

5.4 Implementation Details
Methodology:
The main idea of boosting algorithms is combining many simple and moderately accurate hypotheses called weak classifiers into a single, highly accurate classifier called strong classifier for the task at hand. The weak classifiers are trained sequentially and, conceptually, each of them is trained mostly on the examples which were most difficult to classify, by the preceding weak classifiers.

Weak Classifier Design
The goal of the weak classifier is to find a weak hypothesis with moderately low weighted error (i.e. minimal sum of the weight of misclassified examples). Initially, the distribution is uniform, but the boosting algorithm updates the weights on each round to force the weak learner to concentrate on the pairs (examples, label) which are hardest to predict. So the main idea is to run the weak classifier several times, each time on a different distribution of instances, to generate several different hypotheses. Refer to these hypotheses as the “weak” hypotheses. These weak hypotheses are combined by the boosting algorithm into a single more complex and more accurate hypothesis. The weak classifier selects a ranking feature, and converts it into a binary +1 and -1 by threshold. The threshold feature judges the train data with scores above the threshold as relevant (score of 1) and the train data with scores below the threshold as non-relevant (score of -1) or vice versa.
A group of weak classifiers has to be prepared as input of AdaBoost algorithm. They can be linear classifiers, ANNs or other common classifiers. In this work “decision stumps” is used as weak classifiers due to its simplicity. For every feature $f$, its value range could be divided into two non overlapping value subsets $C^f_p$ and $C^f_n$, and the decision stump on $f$ takes the form as follow:

$$h_f(x) = \begin{cases} +1 & x(f) \in C^f_p \\ -1 & x(f) \in C^f_n \end{cases}$$

(5.1)

Where $x(f)$ indicates the value of $x$ on feature $f$.

To implement the weak learner efficiently, we search for the hypothesis $h$ that minimizes the weighted classification errors. The Weak learner algorithm searches for $h$ by checking all possible thresholds for all features.

The weak learner algorithm should generate a sequence of weak hypotheses (classifiers) with mean low error rate higher than $\frac{1}{2}$, with high probability. For computing of hypothesis error, the weak hypotheses $h$ are comparing to the labels $y$.

**DS-AdaBoost algorithm**

AdaBoost is an ensemble method that constructs a classifier in an iterative fashion. In each iteration, it calls a simple learning algorithm (the weak learner) that returns a classification. The final classification will be decided by a weighted “vote” of the weak classifiers, where each weight is proportional to the correctness of the corresponding weak classifier. This incremental process of combining weak classifiers weighed by their performance is called boosting. Weak classifiers need only be slightly better than a random guess, which lends great flexibility to the design of the weak classifier (or feature) set. If there is no particular a-priori knowledge available on the domain of the learning problem, small decision trees or, in the extreme case, decision stumps (decision trees with two leaves) are often used. A decision stump can be defined by three parameters, the index $j$ of the attribute that it cuts, the threshold $\mu$ of the cut, and the sign of the decision.

In this algorithm, weak classifiers are selected iteratively from a number of candidate weak classifiers and combined linearly to form a strong classifier for classifying the network data. Let $H = \{h_f\}$ be the set of constructed weak classifiers. The set of training sample data be $(x_1, y_1), \ldots, (x_i, y_i), \ldots, (x_n, y_n)$ where $x_i$ denotes the $i^{th}$ feature
vector, \( y_i \in \{+1, -1\} \) is the label of the \( i^{th} \) feature vector, denoting whether the feature vector represents a normal behaviour or not; and \( n \) is the size of the data set.

This algorithm runs for \( T \) rounds. Each sample \( i \) is assigned a weight \( w_i(t) \) at any round \( t \). Initially, all weights are set equally, but during the execution of the algorithm these weights are redistributed in order to manipulate the selection process. In every round \( t \) the performance of each single weak classifier \( h_j \) on all \( n \) training samples is assessed. The performance is measured by the weighted error defined as:

\[
\varepsilon_j = \sum_{i=1}^{n} w_i(t) I[y_i \neq h_j(x_i)]
\]  

(5.2)

Where

\[
I_{[\gamma]} = \begin{cases} 
1, & \gamma = true \\
0, & \gamma = false 
\end{cases}
\]

At the end of each round, the classifier \( h \) with the lowest error rate \( \varepsilon_t \) based on equation 5.2 is selected and stored as best classifier \( h_t \) of round \( t \). Then a confidence \( \alpha_t \) is computed as:

\[
\alpha_t = \frac{1}{2} \log \left( \frac{1 - \varepsilon_t}{\varepsilon_t} \right)
\]

(5.3)

and can be interpreted as the quality of hypothesis \( h_t \); the lower error rate \( \varepsilon_t \), the higher confidence \( \alpha_t \) . Intuitively, \( \alpha_t \) measures the importance that is assigned to \( h_t \). Note that \( \alpha_t > 0 \) if \( \varepsilon_t > \frac{1}{2} \) and that \( \alpha_t \) gets larger as \( \varepsilon_t \) gets smaller. Finally, the weights are redistributed and normalized as follows:

\[
w_i(t + 1) = \left( \frac{w_i(t) \exp(-\alpha_t y_i h_t(x_i))}{Z_t} \right)
\]

(5.4)

By increasing the weights of samples that were misclassified by \( h_t \) favors, in the next round, these difficult samples are handled correctly. The effect of this rule is to increase the weight of examples misclassified by \( h_t \) and to decrease the weight of correctly classified examples. Thus, the weight tends to concentrate on "hard" examples. \( Z_t \) denotes
the normalization factor which ensures the sum of all weights to be +1. The normalization factor is defined as:

$$Z_t = \sum_{k=1}^{n} \exp(-\alpha_t y_i h_t(x_k))$$  \hspace{1cm} (5.5)

The final hypothesis $H$ is a weighted majority vote of the $T$ selected weak classifiers where $\alpha_t$ is the weight assigned to hypothesis $h_t$:

$$H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(x) \right)$$ \hspace{1cm} (5.6)

The Algorithm 1 shows the pseudo code for DS-AdaBoost.

**Algorithm 5.1 : DS – AdaBoost**

Require: $(x_1, y_1), \ldots, (x_n, y_n); \ x_i \in X, y_i \in \{+1, -1\}$

Initialize weights

1. Initialize weights $w_1(i) = \frac{1}{n}$
2. For all $t$ such that $0 \leq t \leq T$ do
3. \hspace{0.5cm} Find the minimum classification error $h_t = \min_{h_j \in H} \varepsilon_j$
4. \hspace{0.5cm} if $\varepsilon_j \geq \frac{1}{2}$ then
5. \hspace{1cm} Reinitialize the weights to $\frac{1}{n}$
6. \hspace{0.5cm} goto step 3
7. \hspace{0.5cm} end if
8. \hspace{0.5cm} Compute the confidence $\alpha_t$
9. \hspace{0.5cm} Update the weights $w_t(t + 1)$
10. \hspace{0.5cm} Normalize weight of each sample
11. end for
12. Output the strong classifier $H(x)$
The main reasons why our DS-AdaBoost algorithm obtains good results for intrusion detection are:

1. By combining the decision stumps for both categorical and continuous features into a strong classifier, the relations between categorical and continuous features are handled naturally, without any forced conversions between continuous and categorical features.

2. For the AdaBoost algorithm, it has been proved that the weighted classification error rate for the strong classifier converges to zero as the number of iterations increases i.e., when

\[
T \to \infty, \quad \sum_{i=1}^{n} w_i^{(1)} I[H(x_i) \neq y_i] \to 0 \tag{5.7}
\]

provided that the misclassification rates for the weak classifiers are less than 50%.

The decision stumps minimize the sum of the false-classification rates for normal and attack data. It is guaranteed that the misclassification rates for the selected weak classifiers are lower than 50% this ensures the convergence of the algorithm [16, 151],

### 5.5 Experimentations

Decision Stump-AdaBoost (DS-AdaBoost) algorithm is an enhancement to the existing AdaBoost algorithm that we have proposed in this research work. Here the decision stump is used with AdaBoost algorithm to improve the performance of the classifier. In the AdaBoost algorithm, overfitting to some weak classifiers can easily occur. The problem of over fitting is handled in our work. In the first \( Tq \) iterations, if the sum of the weighted errors for a weak classifier is less than a threshold \( \theta q \), it is considered that this weak classifier fits the samples over-well, i.e., overfits the samples. Then, we select, from the weak classifiers for each of which the sum of the weighted errors is not less than \( \theta q \), the optimal weak classifier that produces the minimum of the weighted errors compared with other weak classifiers for each of which the sum of the weighted errors is not less than \( \theta q \). After the first \( Tq \) iterations, this correction for overfitting is discarded which help to improve the performance of our algorithm.
Experimentation was carried out for DS-AdaBoost algorithm using two data sets i.e. KDDCup99 and NSL-KDD. For comparison the performance of our supervised DS-AdaBoost algorithm, the other standard algorithms are used. For training and testing we used 10% KDDCup99 training and testing data set. Here we discuss the results of our algorithm on KDDCup 99 data set. Table 5.1 shows the performance of DS-AdaBoost using 10% KDDCup99 training data set.

Table 5.1: Performance on training data set

<table>
<thead>
<tr>
<th></th>
<th>Normal</th>
<th>DOS</th>
<th>R2L</th>
<th>U2R</th>
<th>PROBE</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>97218</td>
<td>19</td>
<td>9</td>
<td>0</td>
<td>32</td>
<td>99.93</td>
</tr>
<tr>
<td>DOS</td>
<td>20</td>
<td>391413</td>
<td>3</td>
<td>4</td>
<td>18</td>
<td>99.98</td>
</tr>
<tr>
<td>R2L</td>
<td>15</td>
<td>0</td>
<td>1102</td>
<td>4</td>
<td>5</td>
<td>98.04</td>
</tr>
<tr>
<td>U2R</td>
<td>5</td>
<td>0</td>
<td>45</td>
<td>2</td>
<td></td>
<td>88.46</td>
</tr>
<tr>
<td>PROBE</td>
<td>40</td>
<td>11</td>
<td>9</td>
<td>0</td>
<td>4047</td>
<td>98.53</td>
</tr>
<tr>
<td>%</td>
<td>99.92</td>
<td>99.99</td>
<td>98.22</td>
<td>85.18</td>
<td>98.58</td>
<td>99.96</td>
</tr>
</tbody>
</table>

When algorithm is applied on the testing data set, the results were obtained as shown in Table 5.2 with confusion matrix.

Table 5.2: Performance on testing data set

<table>
<thead>
<tr>
<th></th>
<th>Normal</th>
<th>DOS</th>
<th>R2L</th>
<th>U2R</th>
<th>PROBE</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>60521</td>
<td>25</td>
<td>10</td>
<td>0</td>
<td>37</td>
<td>99.88</td>
</tr>
<tr>
<td>DOS</td>
<td>2880</td>
<td>226865</td>
<td>2</td>
<td>4</td>
<td>102</td>
<td>98.70</td>
</tr>
<tr>
<td>R2L</td>
<td>167</td>
<td>1</td>
<td>13556</td>
<td>3</td>
<td>54</td>
<td>98.37</td>
</tr>
<tr>
<td>U2R</td>
<td>125</td>
<td>6</td>
<td>2439</td>
<td>62</td>
<td></td>
<td>92.52</td>
</tr>
<tr>
<td>PROBE</td>
<td>30</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>4127</td>
<td>98.53</td>
</tr>
<tr>
<td>%</td>
<td>94.98</td>
<td>99.98</td>
<td>99.84</td>
<td>99.71</td>
<td>94.18</td>
<td>98.86</td>
</tr>
</tbody>
</table>
Table 5.3 shows the False Positive Rate and the Detection Rate for training data set and testing data set with proposed DS-AdaBoost algorithm.

<table>
<thead>
<tr>
<th></th>
<th>Training Set</th>
<th>Testing Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPR (%)</td>
<td>0.06</td>
<td>0.118</td>
</tr>
<tr>
<td>DR (%)</td>
<td>99.7</td>
<td>98.86</td>
</tr>
</tbody>
</table>

The experimental result reveals that our proposed Ds-AdaBoost algorithm performs more accurately with low false alarm rate and high detection rate. More experimentation results and comparison of performance of this algorithm with other algorithms are discussed in chapter 7.

**Summary**

The AdaBoost algorithm is one of the most popular machine learning algorithms. The AdaBoost algorithm corrects the misclassifications made by weak classifiers, and it is less susceptible to over-fitting than most learning algorithms. We have proposed and implemented DS-AdaBoost algorithm and studied the detailed results for various attacks. This algorithm is an enhancement to the existing AdaBoost algorithm that we have proposed in this research work. Here the decision stump is used with AdaBoost algorithm to improve the performance of the classifier. In the AdaBoost algorithm, overfitting to some weak classifiers can easily occur. The problem of over fitting is handled in our work.

We have also studied the comparative accuracy of this algorithm with different intrusion detection models. The experimental result reveals that our algorithm performs more accurately than unsupervised SOM and supervised hybrid DT-SVM algorithms. Also the algorithm shows better results than many other existing algorithms. From this we can say that the performance of our DS-AdaBoost algorithm is much better than other intrusion detection models.