An SVM based Efficient and Fast IDS System using Feature Selection and Sampling Algorithm

**Sheetal Kori1, Anurag Jain2, Rajneesh Pachouri3**

*1Department of CSE, AIST, Sagar (M.P.), India*

*E-mail:* [*sheetalkori656@gmail.com*](mailto:sheetalkori656@gmail.com)

*2,3Assistant Professor, Department of CSE, AIST, Sagar (M.P.), India*

***Abstract***

***IDS systems face challenges such as large feature sets, imbalanced datasets, and a high number of samples, which can affect performance. To address these challenges, this research proposes an SVM-based IDS system that utilizes Fisher's score for feature selection, over- and under-sampling techniques for balancing the dataset, and a probabilistic sample reduction technique to reduce the size of the dataset. Fisher's score is applied to select the most relevant features from a large feature set, while over- and under-sampling techniques balance the dataset by over-sampling the minority class and under-sampling the majority class. Additionally, a probabilistic training data sampling technique is employed to reduce the size of the dataset and, hence, training time and resource requirements. The proposed SVM-based IDS system is evaluated on the KDD99 dataset, and the experimental results demonstrate its superiority over existing SVM-based IDS systems in terms of accuracy, precision, recall, and F1-score.***

***Keywords: Intrusion detection system (IDS), Support vector machine (SVM), KDD99.***

# Introduction

With the increasing dependence of organizations on computer networks and the internet, ensuring the security of these networks has become a critical issue. Intrusion detection systems (IDS) play a vital role in protecting computer networks from various types of attacks, such as denial-of-service (DoS) attacks, malware attacks, and other forms of cybercrime. An IDS is designed to detect suspicious activities on a network and generate alerts to network administrators or security personnel to take appropriate action.

IDS can be categorized into two main types: host-based IDS (HIDS) and network-based IDS (NIDS). HIDS operates at the individual host level, while NIDS monitors network traffic for suspicious activities. Both HIDS and NIDS have their advantages and disadvantages, and organizations often deploy a combination of both to achieve comprehensive security.

One of the key challenges in developing effective IDS systems is achieving high accuracy in detecting malicious activities while minimizing false positives. False positives can be costly and time-consuming to investigate and can lead to unnecessary downtime and resource utilization. Various approaches have been proposed to address this challenge, such as using machine learning algorithms like support vector machines (SVM), decision trees, and neural networks.

SVMs have become a popular choice for developing IDS due to their high accuracy in classifying network traffic. SVM-based IDS systems can be trained to distinguish between normal and anomalous network behavior using labeled training data. Once trained, the SVM-based IDS system can classify incoming network traffic as either normal or anomalous based on the learned patterns. However, SVM-based IDS systems can suffer from imbalanced datasets, a large number of features, and a high number of samples, which can lead to overfitting and reduced performance.

Feature selection, sample balancing, and sample reduction techniques have been proposed to address these challenges. Feature selection aims to identify the most relevant features from a large feature set, while sample balancing is used to address the issue of imbalanced datasets. Sample reduction techniques aim to reduce the size of the dataset by removing redundant or irrelevant samples. These techniques can improve the performance of SVM-based IDS systems by reducing overfitting and enhancing the generalization of the model.

Intrusion detection systems are critical components of computer network security, and SVM-based IDS systems have proven to be effective in detecting and preventing malicious activities. However, SVM-based IDS systems face challenges such as imbalanced datasets, a large number of features, and a high number of samples. Feature selection, sample balancing, and sample reduction techniques have been proposed to address these challenges and improve the performance of SVM-based IDS systems.

The rest of the paper is organized as follows in section 2 a brief review of the relevant work is provided. Section 3 provides the theoretical explanation of the various methods used in the work. The section 4 explains the proposed algorithm followed by the experimental results and analysis in section 5. Finally, the section 6 concludes the paper.

# Literature review

An Intrusion Detection System (IDS) is a security mechanism that monitors network traffic or system activity for signs of security breaches, policy violations, or other suspicious activities. IDS systems are used to detect and respond to security threats and protect computer networks and systems from unauthorized access or attacks [1].

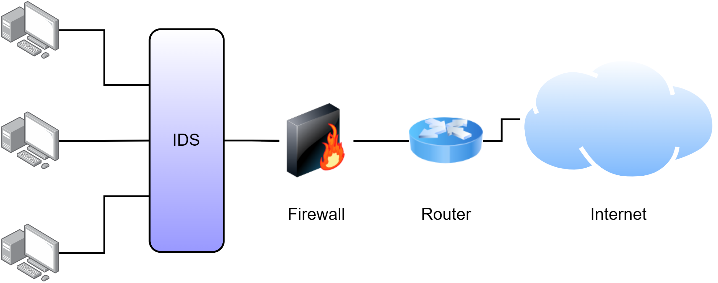


Fig.1. Position of IDS in a network.

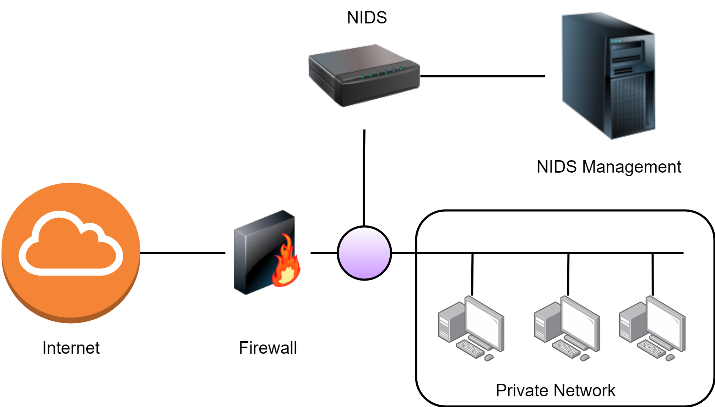


Fig.2. Network-based Intrusion Detection Systems (NIDS).

## Network Based Intrusion Detection System (NIDS)

Network-based IDS monitor network traffic for suspicious activity, looking for signs of an attack such as unusual traffic patterns, suspicious packet headers, or known attack signatures. NIDS operate by inspecting network traffic as it flows through a specific point on the network, such as a network switch or router, and can identify both internal and external attacks. These systems are often used in larger networks and are typically installed at the perimeter of the network. Adam et al. [2], presented a Neural Network Intrusion Detection System (NNIDS) that can detect malicious activities in a network environment. The NNIDS is based on anomaly detection and was tested for various types of attacks. The proposed NNIDS system is able to successfully recognize learned malicious activities in a network environment. Waheed et al. [3], proposed a cyber-intrusion detecting system called HADMLP that uses a hybrid metaheuristic algorithm and feature selection based on a multi-objective wrapper method. Wenkee et al. [4], presented a framework called MADAM ID for developing intrusion detection systems (IDSs) using data mining algorithms. The framework uses system audit data to compute activity patterns and extract predictive features, which are then used to generate intrusion detection rules using machine learning algorithms. David et al. [5], proposed an intrusion detection system (IDS) using machine learning classification technique with the Naive Bayes algorithm. The study focuses on optimizing the performance of the algorithm by applying K-Means Clustering method for continuous variable discretization and feature selection. Zhou et al. [6], proposed a new intrusion detection framework that uses feature selection and ensemble learning techniques to improve the performance of existing intrusion detection algorithms. Alazzam et al. [7], The paper proposes a new algorithm for feature selection in Intrusion Detection System (IDS) using the pigeon inspired optimizer. Rahman et al. [8], proposed two methods, semi-distributed and distributed, to design an intrusion detection system (IDS) for resource-constrained devices in the Internet of Things (IoT) network. These methods combine well-performing feature extraction and selection and exploit potential fog-edge coordinated analytics to overcome the limitations of the centralized IDS. Rajagopal et al. [9], proposed a hybrid multi model solution for network intrusion detection using an ensemble model with meta-classification approach enabled by stacked generalization. Yan et al. [10], proposed a deep learning approach called stacked sparse autoencoder (SSAE) to extract high-level feature representations of intrusive behaviour information for intrusion detection.

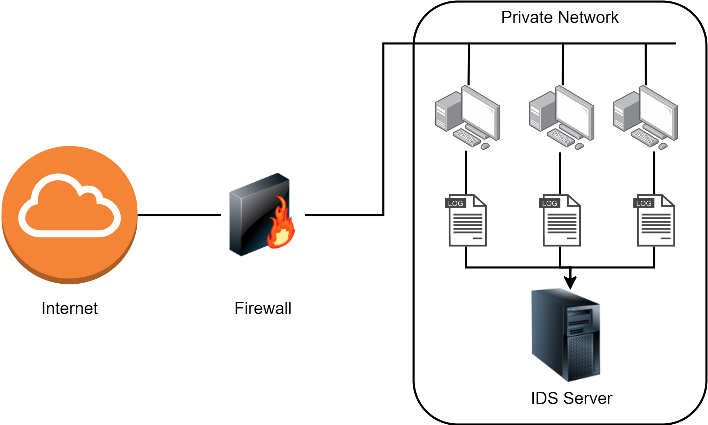


Fig.3. Host-based Intrusion Detection Systems (HIDS).

## Host Based Intrusion Detection System (HIDS)

Host-based IDS monitor the activities on individual hosts, looking for signs of unauthorized access or malicious activity. HIDS monitor system logs, file changes, system calls, and other host activity that may indicate a security threat. These systems are typically installed on individual hosts, such as servers or workstations, and can provide more detailed information about the activities of a specific user or process. Generator et al. [11], discusses the need for a network security system to deal with potential threats that can occur quickly and accurately by utilizing an IDS (intrusion detection system). It highlights the use of Snort, an IDS tool that works in real-time to monitor and detect ongoing network threats, specifically DoS attacks. Shijoe et al. [12], proposed an intrusion detection system (IDS) that monitors network or system activities to detect malicious signs and gives alerts when a user tries to intrude. Subba et al. [13], This paper proposes a new framework for host intrusion detection system (HIDS) that uses machine learning to identify anomalous system processes in real-time. The framework uses a combination of tf-idf vectorizer and truncated singular value decomposition (SVD) to transform system call trace files into n-gram feature vectors and then reduce their dimensionality for efficient processing. Yukyung et al. [14], proposed a method to improve the accuracy of anomaly-based intrusion detection systems by using machine learning algorithms for classification of normal and attack data. Chung-Ming et al. [15], proposes an adaptable agent-based IDS inspired by the danger theory of artificial immune system. The learning mechanism of the proposed IDS is designed by emulating how dendritic cells (DC) in immune systems detect and classify danger signals. Yulia et al. [16], proposes an approach for detecting Botnet attacks using a host-based Intrusion Detection System. The approach is based on a genetic algorithm to detect anomalies during attacks.

# Theoretical Background

In this section different concepts such as dataset balancing, feature selection, classification etc., used in the proposed approach has been discussed and explained.

## Dataset Balancing

Dataset balancing algorithms are used to address the problem of imbalanced data, where the distribution of the classes in the dataset is uneven. This is a common problem in many machine learning tasks, such as fraud detection, disease diagnosis, and anomaly detection, where the minority class is of particular interest, but is often underrepresented in the dataset.

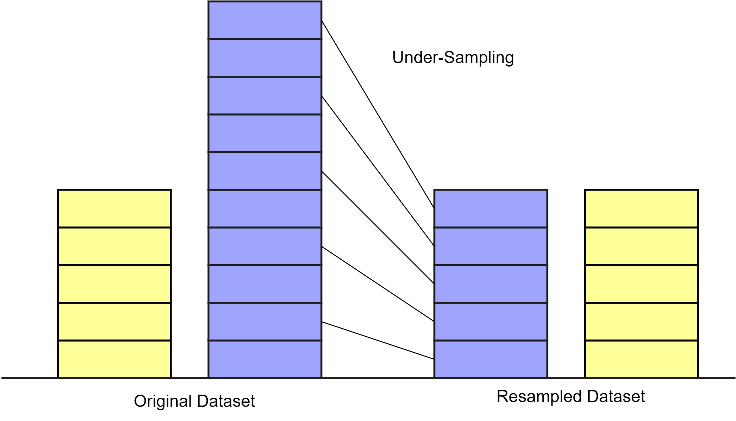


Fig.4. Under-Sampling to reduce the samples of majority class.

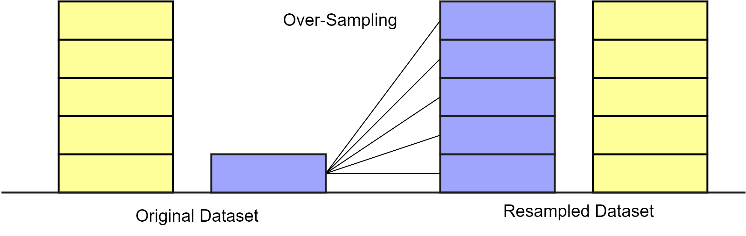


Fig.5. Over-Sampling to increase samples in minority class.

There are several techniques for balancing imbalanced datasets, including:

1. Oversampling: In this technique, the minority class is oversampled by generating synthetic examples from the existing data. This can be done using methods such as SMOTE (Synthetic Minority Over-sampling Technique), ADASYN (Adaptive Synthetic Sampling), and Random Oversampling.
2. Under sampling: In this technique, the majority class is under sampled by randomly removing examples from the dataset. This can be done using methods such as Random Under sampling, Tomek Links, and NearMiss.

## Fisher Feature Selection Algorithm

The Fisher feature selection algorithm is a popular method for selecting relevant features in binary classification problems. It aims to identify the features that have the greatest discriminating power between the two classes, based on their statistical properties. The algorithm uses the Fisher criterion, which is defined as the ratio of the between-class variance to the within-class variance, to measure the separability of the classes. Here is a step-by-step explanation of the Fisher feature selection algorithm, along with the relevant mathematical equations:

1. Calculate the mean and variance of each feature for each class. For a binary classification problem with two classes (0 and 1), let be the matrix of data points with features. Let and be the mean vectors of the features for class and class , respectively. Let and be the covariance matrices of the features for class and class , respectively. Then, we can calculate the mean and variance of each feature for each class as follows:

|  |  |  |
| --- | --- | --- |
|  |  | () |
|  |  | () |

where and are the number of data points in class and class , respectively.

1. Calculate the Fisher criterion for each feature. The Fisher criterion for a feature is defined as the ratio of the between-class variance to the within-class variance, and is given by:

|  |  |  |
| --- | --- | --- |
|  |  | () |

1. Rank the features based on their Fisher scores. Sort the features in descending order of their Fisher scores, and select the top features as the relevant features.

The Fisher feature selection algorithm aims to identify the features that have the greatest discriminating power between the two classes, based on their statistical properties. By calculating the Fisher criterion for each feature, we can measure the separability of the classes and select the top features with the highest Fisher scores. This approach is widely used in machine learning and data analysis applications to reduce the dimensionality of the data and improve the accuracy of the classification models.

## Support Vector Machine (SVM)

Support Vector Machines (SVMs) are a popular type of supervised learning algorithm used for classification and regression tasks. The main idea behind SVMs is to find the hyperplane that maximally separates the classes of the input data points in a high-dimensional space. The hyperplane is chosen in such a way that it maximizes the margin, which is the distance between the hyperplane and the closest data points of each class. Here is a detailed explanation of SVMs, including the key concepts and mathematical equations involved:

1. **Hyperplane:** In a binary classification problem, the hyperplane is a line that separates the two classes of data points. In a multi-class classification problem, the hyperplane is a higher-dimensional surface that separates the classes. The hyperplane is defined by the equation:

|  |  |  |
| --- | --- | --- |
|  |  | () |

where is the weight vector that determines the orientation of the hyperplane, is the input data point, is the bias term that determines the position of the hyperplane, and denotes the transpose of the vector.

1. **Margin:** The margin is the distance between the hyperplane and the closest data points of each class. The margin is given by the formula:

|  |  |  |
| --- | --- | --- |
|  |  | () |

where is the Euclidean norm of the weight vector .

1. **Support Vectors:** The support vectors are the data points that are closest to the hyperplane, and they determine the position of the hyperplane. The support vectors are the only data points that need to be stored in memory, which makes SVMs memory-efficient for large datasets.
2. **Soft Margin:** SVM In cases where the classes are not perfectly separable by a hyperplane, we can use a soft margin SVM. A soft margin SVM allows for some data points to be misclassified, and introduces a penalty term for misclassifications in the objective function. The objective function for a soft margin SVM is given by:

|  |  |  |
| --- | --- | --- |
|  | subject to | () |

where is the penalty parameter that controls the trade-off between maximizing the margin and minimizing the misclassification error, is the slack variable that allows for misclassifications, and is the class label of the ith data point.

1. **Kernel Trick:** In cases where the input data is not linearly separable, we can use a kernel trick to map the input data to a higher-dimensional space where it is linearly separable. A kernel function is used to compute the dot product of the mapped feature vectors, without explicitly computing the mapping. Some popular kernel functions include the linear kernel, polynomial kernel, and Gaussian radial basis function (RBF) kernel.

SVMs are a powerful and flexible machine learning algorithm that can handle both linearly separable and non-linearly separable data, and can be extended to handle multi-class classification and regression tasks. SVMs are widely used in various applications, such as image recognition, text classification, and bioinformatics.

### SVM for Multiclass Classification

SVMs are originally designed for binary classification problems, where the goal is to separate two classes of data points using a hyperplane. However, SVMs can be extended to handle multiclass classification problems, where the goal is to classify data points into more than two classes. There are two main approaches for multiclass classification using SVMs: One-vs-All, and One-vs-One.

1. **One-vs-All:** In this approach, we train different binary SVM classifiers, where is the number of classes in the dataset. Each classifier is trained to separate one of the classes from the rest of the data. During testing, we apply all classifiers to a test data point and choose the class with the highest classifier score as the predicted class. The classifier score is the signed distance between the test data point and the hyperplane of the corresponding SVM classifier.

The One-vs-All approach is simple to implement and efficient for large datasets, but it may lead to imbalanced class distributions and unstable classification results.

Table.1. NSL-KDD dataset class distribution.

|  |  |  |  |
| --- | --- | --- | --- |
| **Category** | **Training Dataset** | **Testing Dataset** | |
| **KDDTrain+\_20Percent** | **KDDTest+** | **KDDTest-21** |
| Normal | 13449 | 9711 | 2152 |
| Probe | 2289 | 2421 | 2402 |
| DoS | 9234 | 7458 | 4342 |
| U2R | 11 | 200 | 200 |
| R2L | 209 | 2758 | 2754 |
| Total | 25192 | 22544 | 11850 |

1. **One-vs-One:** In this approach, we train different binary SVM classifiers, where is the number of classes in the dataset. Each classifier is trained to separate a pair of classes from the data. During testing, we apply all classifiers to a test data point and count the number of votes for each class. The class with the highest number of votes is chosen as the predicted class.

The One-vs-One approach is more computationally expensive than the One-vs-All approach, but it is less sensitive to imbalanced class distributions and can handle more complex decision boundaries.

To summarize, SVMs can be extended to handle multiclass classification problems using either the One-vs-All or One-vs-One approach. Both approaches have their advantages and disadvantages, and the choice of approach depends on the specific problem and dataset.

## Dataset

NSL-KDD is a benchmark dataset for evaluating intrusion detection systems (IDSs) that was developed as an improvement over the KDD Cup 1999 dataset. The KDD99 dataset is a widely used dataset for evaluating IDSs, but it has several limitations, including outdated attack types and an imbalanced class distribution. The NSL-KDD dataset was developed to address these limitations and provide a more challenging and realistic benchmark dataset for evaluating IDSs.

The NSL-KDD dataset includes a selection of network traffic data from the original KDD Cup 1999 dataset, but with several improvements. Specifically, the NSL-KDD dataset includes:

1. **Elimination of redundant and irrelevant features:** The original KDD Cup 1999 dataset included many redundant and irrelevant features, which can negatively impact the performance of IDSs. The NSL-KDD dataset includes a reduced set of 41 features, which have been carefully selected to be relevant and non-redundant.
2. **Updated attack types:** The original KDD Cup 1999 dataset included attack types that were no longer relevant or realistic for modern networks. The NSL-KDD dataset includes new attack types that are more realistic and relevant for modern networks, such as web-based attacks and botnet attacks.
3. **Balanced class distribution:** The original KDD Cup 1999 dataset had an imbalanced class distribution, with a majority of normal instances and only a small number of anomalous instances. The NSL-KDD dataset has a more balanced class distribution, with 42.51% normal instances and 57.49% anomalous instances.

### Attack Types

The NSL-KDD dataset includes four main categories of attacks, which are:

1. **Denial of Service (DoS):** Attacks that attempt to make a server or network resource unavailable to users by overwhelming the system with a flood of traffic.
2. **User to Root (U2R):** Attacks that exploit vulnerabilities in a system to gain root-level access and take control of the system.
3. **Remote to Local (R2L):** Attacks that attempt to gain unauthorized access to a system from a remote location.
4. **Probe:** Attacks that attempt to gather information about a system's vulnerabilities and weaknesses.

### Features Description

The NSL-KDD dataset includes a total of 41 features, which have been carefully selected to be relevant and non-redundant for evaluating intrusion detection systems (IDSs). The features can be divided into four main categories, which are:

1. **Basic features:** These include features related to basic properties of the network traffic, such as the protocol type, service type, source and destination IP addresses, source and destination port numbers, and the duration of the connection.
2. **Content features:** These include features related to the content of the network traffic, such as the number of failed login attempts, the number of shell prompts, and the number of root accesses.
3. **Traffic features:** These include features related to the volume and pattern of the network traffic, such as the number of bytes and packets sent and received, and the time between packets.
4. **Host-based features:** These include features related to the behavior of the host computer, such as the number of file creations and deletions, and the number of processes running.

# Proposed Algorithm

The functional block diagram of the proposed algorithm is presented in the Fig. 7.1, based on provided block diagram the functionality of the proposed algorithm can be described as follows:

1. Dataset: In the first step the IDS dataset NSL-KDD is collected read for further processing.
2. Pre-processing: this involves following steps to prepare the data for use for further processing:
3. Data cleaning: The dataset may contain missing or erroneous data that needs to be cleaned or removed before analysis.
4. Data normalization: It's important to ensure that the dataset has a consistent scale and range for all features. This is typically achieved through normalization or scaling techniques, such as MinMax scaling or z-score normalization.
5. Handling categorical features: The dataset contains categorical features that need to be converted into numerical values for use in machine learning algorithms. This can be done through one-hot encoding, label encoding, or other techniques.
6. Train Test Split: this is a common technique used in machine learning to split a dataset into two parts: a training set and a testing set. The training set is used to train a machine learning model, while the testing set is used to evaluate its performance. After the data has been split, the training set can be used to train a machine learning model, while the testing set can be used to evaluate its performance. By using a separate testing set, we can get a more accurate estimate of the model's performance on new, unseen data.
7. Group Samples by Classes: To group samples by classes, we simply need to identify the class label for each sample in the dataset and group the samples accordingly. By grouping the samples in the dataset by their class labels, we can better understand the distribution of the data and ensure that each class is represented in the training and testing sets.
8. Mean Data Samples Per Class: The mean data samples per class is a measure of the average number of data samples in a dataset for each class label. This measure can be useful in understanding the distribution of data across different classes, and in ensuring that the training and testing sets contain a representative sample of each class. To calculate the mean data samples per class, we simply count the number of samples in the dataset for each class label and divide by the number of classes. For example, in the NSL-KDD dataset, there are five class labels: normal, dos, probe, r2l, and u2r. To calculate the mean data samples per class for this dataset, we can count the number of samples for each class and divide by five.
9. Compare Per Class Samples with Mean Per Class Samples: the comparison is performed to identify which classes requires over sampling and which requires under sampling.
10. Perform Under Sampling: the classes which contains more samples than the average samples per class in the dataset are under sampled to reduce skewness of the samples distribution and hence to balance the dataset.
11. Perform Over Sampling: the classes which contains less samples than the average samples per class in the dataset are over sampled to reduce skewness of the samples distribution and hence to balance the dataset.
12. Collect All Classes Samples: once all classes are balanced after the over and under sampling the new sampled data from all classes are collected for further processing.
13. Fisher Feature Selection: this is a popular method for selecting a subset of the most informative features in a dataset for classification. The goal of this step is to select a subset of features that maximizes the separation between the different classes in the dataset. In the NSL-KDD dataset, which contains a large number of features, applying a feature selection technique like Fisher feature selection can help improve the performance of a machine learning model.
14. Probabilistic Sampling: it is a technique used to reduce the training sample size in a dataset by selecting a representative subset of samples for training. This can be particularly useful in large datasets where training a machine learning model on the full dataset may be computationally expensive or impractical.
15. SVM Training: intrinsically the SVM is a binary classifier, therefore training an SVM for multiclass classification requires different approach. In this work we adopted one-vs-one approach which involves training a binary SVM classifier for each pair of classes in the dataset. For example, if there are K classes in the dataset, we need to train K\*(K-1)/2 binary SVM classifiers. The basic idea is that for a given pair of classes, we create a new dataset that contains only the samples from those classes. We then train a binary SVM classifier on this new dataset, where one class is treated as the positive class and the other as the negative class. This process is repeated for every pair of classes in the dataset. To make a prediction for a new sample, we first use each binary classifier to classify the sample as belonging to one of the two classes in the pair. We then tally up the number of times each class was predicted and assign the new sample to the class that received the most votes.
16. Evaluation Metrics: Intrusion Detection Systems (IDS) are used to detect and prevent malicious activities on computer networks. The effectiveness of an IDS can be evaluated using several performance metrics, including Accuracy, Precision, Recall, F-Score.

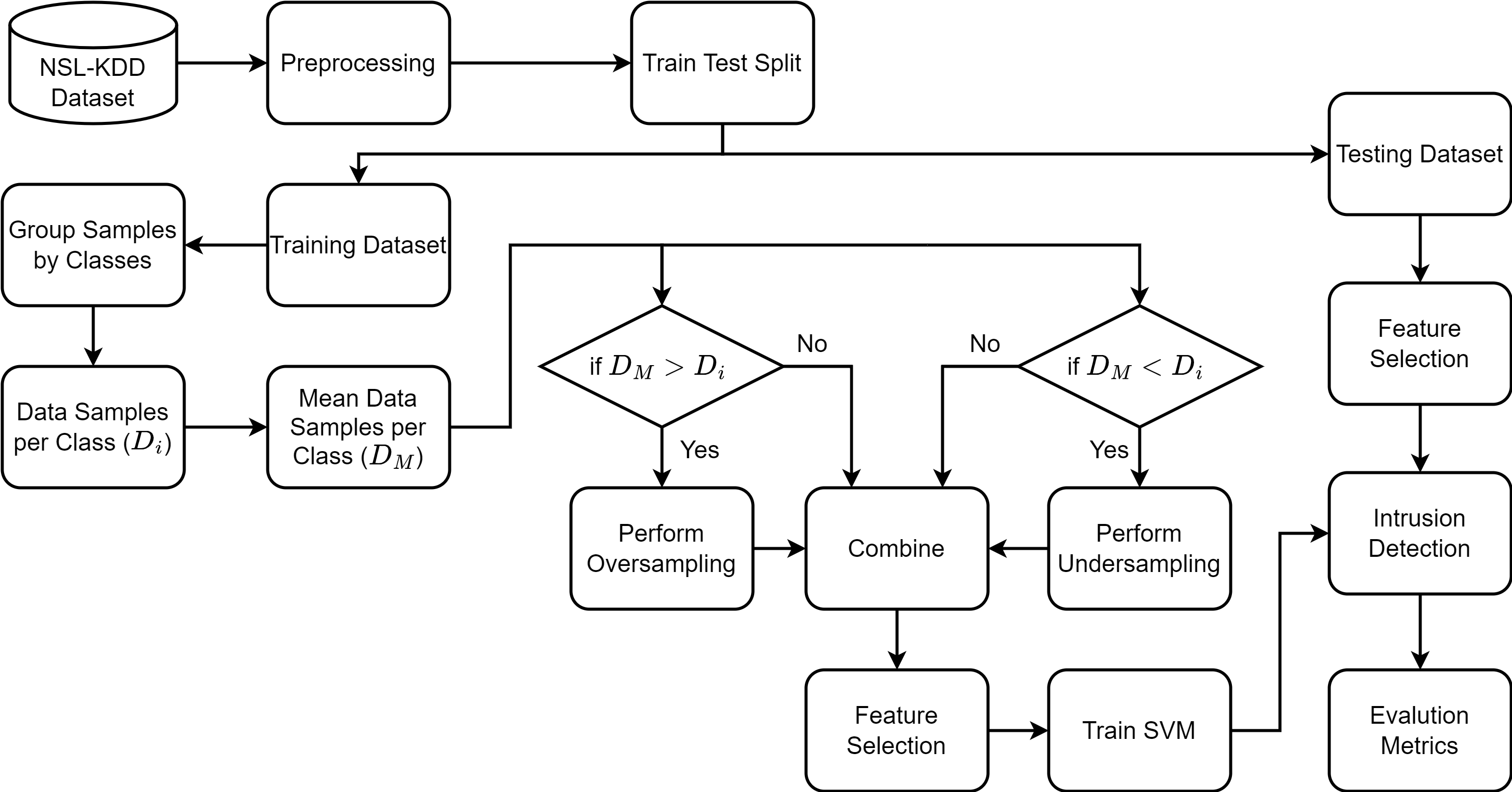


Figure 6: Accenture of the IDS Proposed Algorithm.

### Performance Evolution Metrics

Intrusion Detection Systems (IDS) are used to detect and prevent malicious activities on computer networks. The effectiveness of an IDS can be evaluated using several performance metrics, including:

1. Accuracy: The proportion of all predictions that are correct. It is calculated as (TP + TN) / (TP + TN + FP + FN), where TP is true positive, TN is true negative, FP is false positive, and FN is false negative.
2. Precision: The proportion of true positive predictions out of all positive predictions. It is calculated as TP / (TP + FP).
3. Recall (also called Sensitivity): The proportion of true positive predictions out of all actual positive instances. It is calculated as TP / (TP + FN).
4. F1 Score: The harmonic mean of precision and recall. It is calculated as 2 \* (precision \* recall) / (precision + recall).

Where TP, TN, FP, and FN, denoting the true positive, true negative, false positive and false negative respectively.

By default, all these measures treat all classes equally and assumes that each class has equal importance. However, considering the uneven class distribution in the NSL-KDD dataset. We used weighted version of these measures which assigns different weights to each class based on their prevalence or importance. This means that measure of each class is weighted by the proportion of instances belonging to that class as shown in following equation:

|  |  |
| --- | --- |
|  | () |

where:

, , ..., are the measures of each class.

, , ..., are the weights assigned to each class.

Weighted measure provides a more balanced evaluation when dealing with imbalanced datasets, where certain classes may be underrepresented. By assigning higher weights to minority classes, weighted accuracy gives them more influence in the overall performance assessment.

These metrics can be used to evaluate the performance of different IDS algorithms and to compare the performance of different IDS systems. It is important to note that the choice of metric depends on the specific requirements of the IDS and the nature of the data being analyzed.

# Experimental Results

The simulation of the proposed algorithm is performed using Python on Intel core i5 processor-based PC with 8GB of RAM. The results are calculated for the different size of datasets and with different features. The results are also compared with the KNN, Navie Bayes, MLP, and SVM based classification technique. Following Classification Performance Measures are taken for the analysis and comparison of the algorithms There are many measures available for judgment of the quality of the classifier and all of them are derived from the following confusion matrix.

## Scenario 1:

Binary Classification: where normal samples is considered as class 1, and all other classes ('DoS', 'Probe', 'Privilege', 'Access') samples are considering as class 2.

Table.2. Sample distributions original for scenario 1

|  |  |  |
| --- | --- | --- |
|  | Training | Testing |
| Class 1 | 67351 | 9855 |
| Class 2 | 58621 | 12688 |
| Features | 41 | 41 |

Table.3. Sample distributions after resampling and feature selection for scenario 1.

|  |  |  |
| --- | --- | --- |
|  | Training | Testing |
| Class 1 | 12597 | 9855 |
| Class 2 | 11724 | 12688 |
| Features | 5 | 5 |

Table.4. Evaluation results for scenario 1 (all measures are weighted).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | F1-Score | Training Time | Detection Time |
| KNN | 0.730604 | 0.820064 | 0.730604 | 0.721727 | 0.0689578 | 93.1771 |
| NB | 0.400479 | 0.410038 | 0.400479 | 0.360057 | 0.462656 | 0.113922 |
| MLP | 0.790445 | 0.811978 | 0.790445 | 0.79071 | 64.1064 | 0.0649614 |
| SVM | 0.193497 | 0.179266 | 0.193497 | 0.178013 | 13.2317 | 1.40345 |
| Proposed | 0.845983 | 0.8475 | 0.845983 | 0.844618 | 0.725051 | 0.42873 |

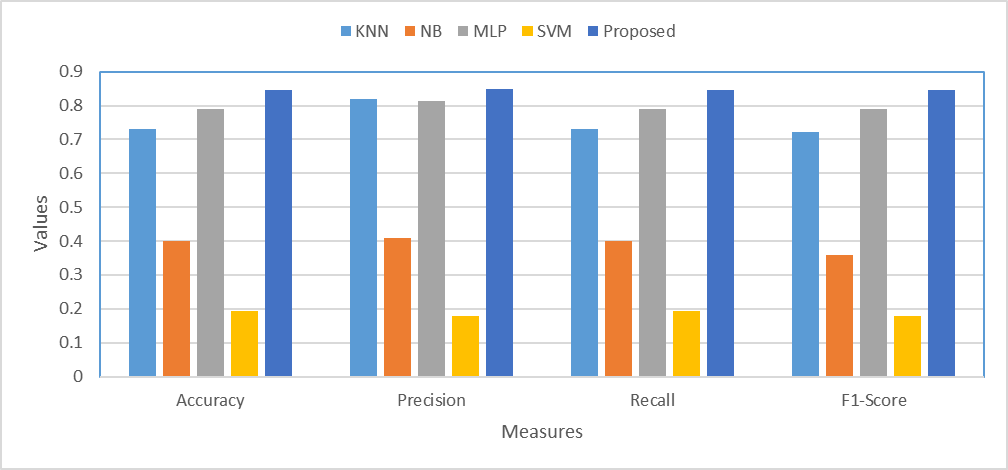


Fig.7. Plots for evaluation results presented in table 4.

|  |  |
| --- | --- |
|  |  |
| (a) KNN | (b) NB |
|  |  |
| (c) MLP | (d) SVM |
|  | |
| (e) Proposed | |
|  | |

Fig.8. Confusion matrix for different classification algorithms for scenario 1.

## Scenario 2:

Binary Classification: where normal samples is considered as class 1, and samples from class 'DoS' are considering as class 2 all other class samples are removed.

Table.5. Sample distributions original for scenario 2.

|  |  |  |
| --- | --- | --- |
|  | Training | Testing |
| Class 1 | 67351 | 9855 |
| Class 2 | 45927 | 7459 |
| Features | 41 | 41 |

Table.6. Sample distributions after resampling and feature selection for scenario 2.

|  |  |  |
| --- | --- | --- |
|  | Training | Testing |
| Class 1 | 11328 | 9855 |
| Class 2 | 9185 | 7459 |
| Features | 5 | 5 |

Table.7. Evaluation results for scenario 2 (all measures are weighted).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | F1-Score | Training Time | Detection Time |
| KNN | 0.880732 | 0.895238 | 0.880732 | 0.877579 | 0.0469706 | 41.5757 |
| NB | 0.644854 | 0.779087 | 0.644854 | 0.562969 | 0.473704 | 0.0889463 |
| MLP | 0.869701 | 0.86977 | 0.869701 | 0.869126 | 37.739 | 0.0559657 |
| SVM | 0.854453 | 0.875414 | 0.854453 | 0.849179 | 10.3984 | 1.18106 |
| Proposed | 0.898117 | 0.906552 | 0.898117 | 0.898605 | 0.599652 | 0.404327 |

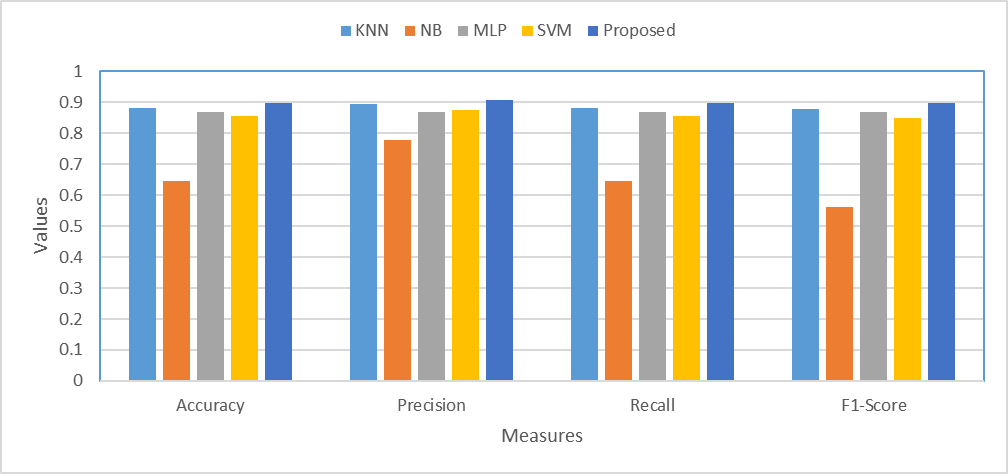


Fig.9. Plots for evaluation results presented in table 7.

|  |  |
| --- | --- |
|  |  |
| (a) KNN | (b) NB |
|  |  |
| (c) MLP | (d) SVM |
|  | |
| (e) Proposed | |
|  | |

Fig.10. Confusion matrix for different classification algorithms for scenario 2.

## Scenario 3:

Binary Classification: where normal samples is considered as class 1, and samples from class “Probe” are considering as class 2 all other class samples are removed.

Table.8. Sample distributions original for scenario 3*.*

|  |  |  |
| --- | --- | --- |
|  | Training | Testing |
| Class 1 | 67351 | 9855 |
| Class 2 | 11656 | 2421 |
| Features | 41 | 41 |

Table.9. Sample distributions after resampling and feature selection for scenario 3.

|  |  |  |
| --- | --- | --- |
|  | Training | Testing |
| Class 1 | 7901 | 9855 |
| Class 2 | 2331 | 2421 |
| Features | 5 | 5 |

Table.10. Evaluation results for scenario 3 (all measures are weighted).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | F1-Score | Training Time | Detection Time |
| KNN | 0.838791 | 0.835757 | 0.838791 | 0.800597 | 0.0229969 | 9.23326 |
| NB | 0.618524 | 0.663693 | 0.618524 | 0.638926 | 0.214867 | 0.0389769 |
| MLP | 0.865754 | 0.874997 | 0.865754 | 0.869353 | 44.871 | 0.0459738 |
| SVM | 0.750163 | 0.68092 | 0.750163 | 0.708318 | 9.30323 | 0.772517 |
| Proposed | 0.875122 | 0.887299 | 0.875122 | 0.879406 | 0.291818 | 0.271832 |

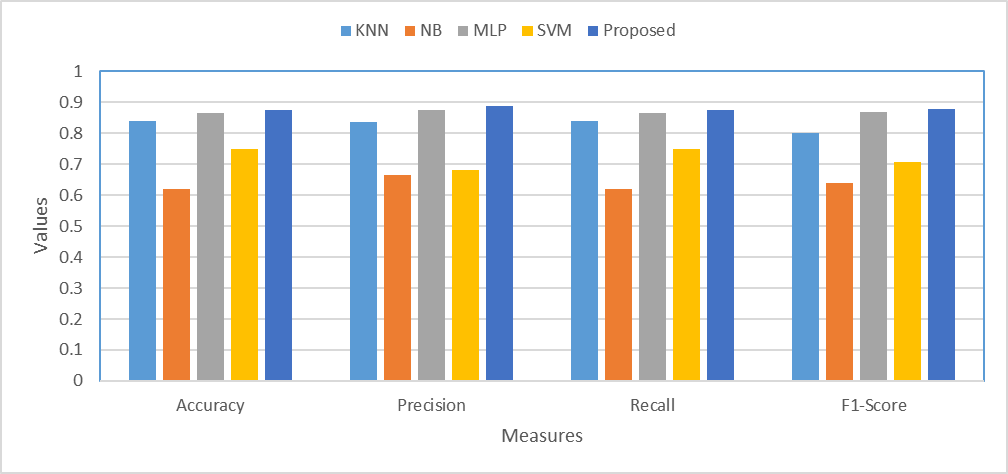


Fig.11. Plots for evaluation results presented in table 10.

|  |  |
| --- | --- |
|  |  |
| (a) KNN | (b) NB |
|  |  |
| (c) MLP | (d) SVM |
|  | |
| (e) Proposed | |

Fig.12. Confusion matrix for different classification algorithms for scenario 3.

## Scenario 4:

Binary Classification: where normal samples is considered as class 1, and samples from class ‘Privilege’ are considering as class 2 all other class samples are removed.

Table.11. Sample distributions original for scenario 4.

|  |  |  |
| --- | --- | --- |
|  | Training | Testing |
| Class 1 | 67351 | 9855 |
| Class 2 | 43 | 65 |
| Features | 41 | 41 |

Table.12. Sample distributions after resampling and feature selection for scenario 4.

|  |  |  |
| --- | --- | --- |
|  | Training | Testing |
| Class 1 | 6739 | 9855 |
| Class 2 | 337 | 65 |
| Features | 5 | 5 |

Table.13. Evaluation results for scenario 4 (all measures are weighted).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | F1-Score | Training Time | Detection Time |
| KNN | 0.993448 | 0.986938 | 0.993448 | 0.990182 | 0.0119967 | 11.6028 |
| NB | 0.95998 | 0.988761 | 0.95998 | 0.973644 | 0.272829 | 0.052969 |
| MLP | 0.993448 | 0.986938 | 0.993448 | 0.990182 | 28.8721 | 0.0229857 |
| SVM | 0.993448 | 0.986938 | 0.993448 | 0.990182 | 1.7779 | 0.160899 |
| Proposed | 0.992137 | 0.993405 | 0.992137 | 0.992702 | 0.206871 | 0.136732 |

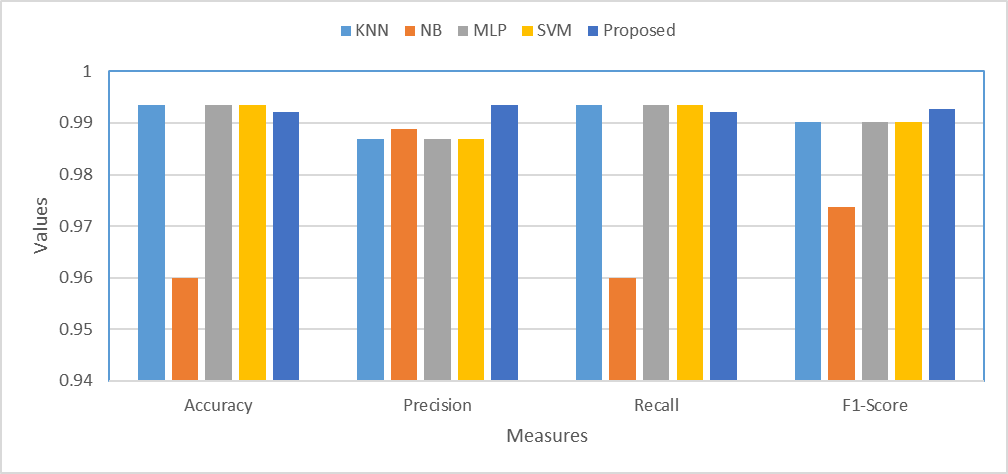


Fig.13. Plots for evaluation results presented in table 13.

|  |  |
| --- | --- |
|  |  |
| (a) KNN | (b) NB |
|  |  |
| (c) MLP | (d) SVM |
|  | |
| (e) Proposed | |

Fig.14. Confusion matrix for different classification algorithms for scenario 4.

## Scenario 5:

Binary Classification: where normal samples is considered as class 1, and samples from class ‘Access’ are considering as class 2 all other class samples are removed.

Table.14. Sample distributions original for scenario 5.

|  |  |  |
| --- | --- | --- |
|  | Training | Testing |
| Class 1 | 67351 | 9855 |
| Class 2 | 995 | 2743 |
| Features | 41 | 41 |

Table.15. Sample distributions after resampling and feature selection for scenario 5*.*

|  |  |  |
| --- | --- | --- |
|  | Training | Testing |
| Class 1 | 6834 | 9855 |
| Class 2 | 352 | 2743 |
| Features | 5 | 5 |

Table.16. Evaluation results for scenario 5 (all measures are weighted).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | F1-Score | Training Time | Detection Time |
| KNN | 0.791237 | 0.828161 | 0.791237 | 0.70793 | 0.019989 | 9.23927 |
| NB | 0.331799 | 0.567576 | 0.331799 | 0.363595 | 0.169897 | 0.0409718 |
| MLP | 0.782743 | 0.829966 | 0.782743 | 0.687834 | 49.4642 | 0.0359774 |
| SVM | 0.792666 | 0.758032 | 0.792666 | 0.743918 | 7.13871 | 1.21876 |
| Proposed | 0.784331 | 0.816495 | 0.784331 | 0.691877 | 0.693107 | 0.384777 |

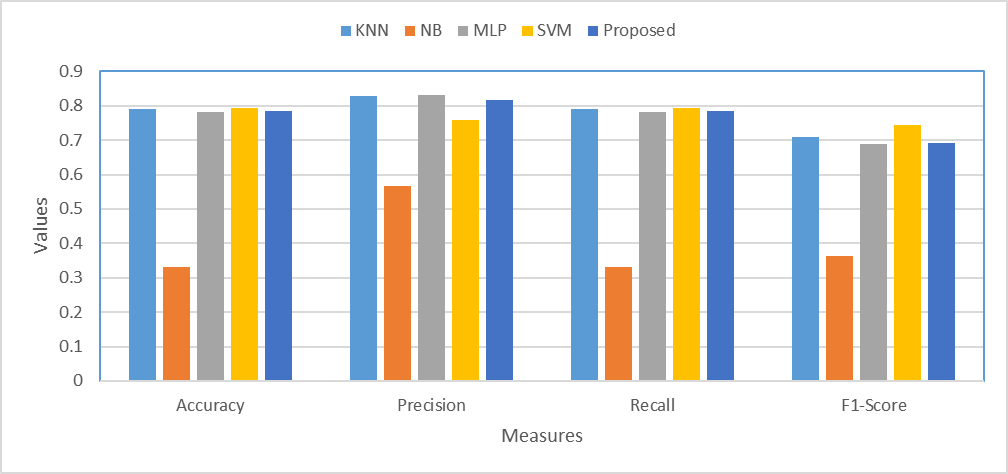


Fig.15. Plots for evaluation results presented in table 16.

|  |  |
| --- | --- |
|  |  |
| (a) KNN | (b) NB |
|  |  |
| (c) MLP | (d) SVM |
|  | |
| (e) Proposed | |

Fig.16. Confusion matrix for different classification algorithms for scenario 5.

## Scenario 6:

Multiclass Classification: all classes ('Normal' ,'DoS', 'Probe', 'Privilege', 'Access') are considered.

Table.17. Sample distributions original for scenario 6.

|  |  |  |
| --- | --- | --- |
|  | Training | Testing |
| Class 1 | 67351 | 9855 |
| Class 2 | 45927 | 7459 |
| Class 3 | 11656 | 2421 |
| Class 4 | 43 | 65 |
| Class 5 | 995 | 2743 |
| Features | 41 | 41 |

Table.18. Sample distributions after resampling and feature selection for scenario 6.

|  |  |  |
| --- | --- | --- |
|  | Training | Testing |
| Class 1 | 5039 | 9855 |
| Class 2 | 5039 | 7459 |
| Class 3 | 2331 | 2421 |
| Class 4 | 252 | 65 |
| Class 5 | 252 | 2743 |
| Features | 30 | 30 |

Table.19. Evaluation results for scenario 6 (all measures are weighted).

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | Accuracy | Precision | Recall | F1-Score | Training Time | Detection Time |
| KNN | 0.689527 | 0.684189 | 0.689527 | 0.625866 | 0.0199964 | 30.1614 |
| NB | 0.214346 | 0.42294 | 0.214346 | 0.235211 | 0.63061 | 0.214866 |
| MLP | 0.710997 | 0.637278 | 0.710997 | 0.668403 | 59.874 | 0.0839472 |
| SVM | 0.509471 | 0.563444 | 0.509471 | 0.459243 | 29.9957 | 8.36081 |
| Proposed | 0.738289 | 0.805659 | 0.738289 | 0.740365 | 1.78489 | 0.329801 |

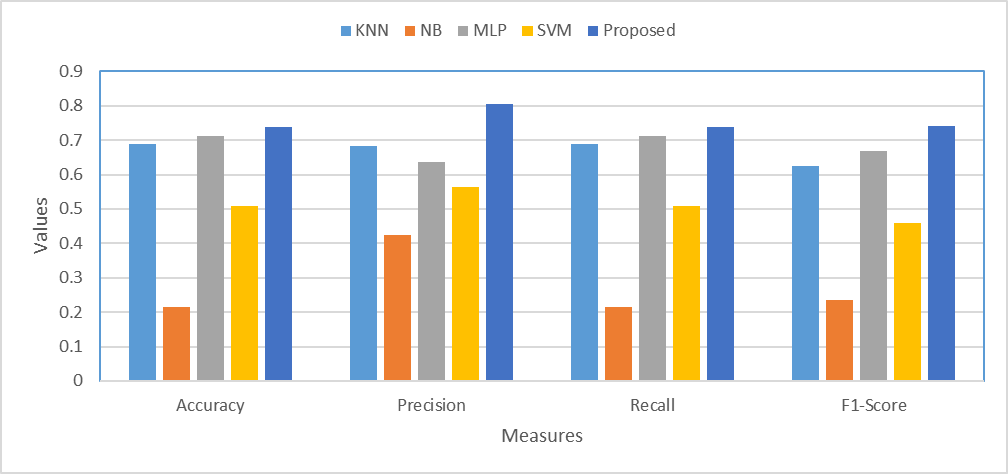


Fig.17. Plots for evaluation results presented in table 19.

|  |  |
| --- | --- |
|  |  |
| (a) KNN | (b) NB |
|  |  |
| (c) MLP | (d) SVM |
|  | |
| (e) Proposed | |

Fig.18. Confusion matrix for different classification algorithms for scenario 6.

# Conclusion

Intrusion detection systems (IDSs) play an important role in computer security. IDS users relying on the IDS to protect their computers and networks demand that an IDS provides reliable and continuous detection service. The model of the Intrusion detector is presented in this thesis is not only capable of detecting the attack situation but can also classifying the individual attacks. The detection accuracy of the system is up to 84.5% (Table-4) while the classification accuracy for each individual attacks is up to 73.8% (Table-19) which are better than all other compared algorithms. The feature selection and sampling reduce the feature and data size and hence also reduces the training and detection time of the proposed algorithm. Which can also be seen from the results that it takes only 0.42 seconds (Table-4) to identify the intrusion compared the SVM which takes 1.4 seconds hence fast enough to prevent any loss due to delayed action. Further it could achieve much better performance by increasing the number of samples taken and increasing the number of characteristics parameter selected.

# References

[1] R. Bace and P. Mell, “NIST Special Publication on Intrusion Detection Systems”.

[2] N. Ádám, B. Madoš, A. Baláž, and T. Pavlik, “Artificial neural network based IDS,” in *2017 IEEE 15th International Symposium on Applied Machine Intelligence and Informatics (SAMI)*, Jan. 2017, pp. 000159–000164. doi: 10.1109/SAMI.2017.7880294.

[3] W. A. H. M. Ghanem *et al.*, “Metaheuristic Based IDS Using Multi-objective Wrapper Feature Selection and Neural Network Classification,” in *Advances in Cyber Security*, M. Anbar, N. Abdullah, and S. Manickam, Eds., in Communications in Computer and Information Science. Singapore: Springer, 2021, pp. 384–401. doi: 10.1007/978-981-33-6835-4\_26.

[4] W. Lee and S. J. Stolfo, “A framework for constructing features and models for intrusion detection systems,” *ACM Trans. Inf. Syst. Secur.*, vol. 3, no. 4, pp. 227–261, Nov. 2000, doi: 10.1145/382912.382914.

[5] D. A. Effendy, K. Kusrini, and S. Sudarmawan, “Classification of intrusion detection system (IDS) based on computer network,” in *2017 2nd International conferences on Information Technology, Information Systems and Electrical Engineering (ICITISEE)*, Nov. 2017, pp. 90–94. doi: 10.1109/ICITISEE.2017.8285566.

[6] Y. Zhou, G. Cheng, S. Jiang, and M. Dai, “Building an Efficient Intrusion Detection System Based on Feature Selection and Ensemble Classifier,” *Computer Networks*, vol. 174, p. 107247, Jun. 2020, doi: 10.1016/j.comnet.2020.107247.

[7] H. Alazzam, A. Sharieh, and K. E. Sabri, “A feature selection algorithm for intrusion detection system based on Pigeon Inspired Optimizer,” *Expert Systems with Applications*, vol. 148, p. 113249, Jun. 2020, doi: 10.1016/j.eswa.2020.113249.

[8] M. A. Rahman, A. T. Asyhari, L. S. Leong, G. B. Satrya, M. Hai Tao, and M. F. Zolkipli, “Scalable machine learning-based intrusion detection system for IoT-enabled smart cities,” *Sustainable Cities and Society*, vol. 61, p. 102324, Oct. 2020, doi: 10.1016/j.scs.2020.102324.

[9] S. Rajagopal, P. P. Kundapur, and K. S. Hareesha, “A Stacking Ensemble for Network Intrusion Detection Using Heterogeneous Datasets,” *Security and Communication Networks*, vol. 2020, p. e4586875, Jan. 2020, doi: 10.1155/2020/4586875.

[10] B. Yan and G. Han, “Effective Feature Extraction via Stacked Sparse Autoencoder to Improve Intrusion Detection System,” *IEEE Access*, vol. 6, pp. 41238–41248, 2018, doi: 10.1109/ACCESS.2018.2858277.

[11] metatags generator, “Intruder Detection Systems on Computer Networks Using Host Based Intrusion Detection System Techniques | Buletin Ilmiah Sarjana Teknik Elektro,” Nov. 2021, Accessed: May 03, 2023. [Online]. Available: http://journal2.uad.ac.id/index.php/biste/article/view/1752

[12] S. Jose, D. Malathi, B. Reddy, and D. Jayaseeli, “A Survey on Anomaly Based Host Intrusion Detection System,” *J. Phys.: Conf. Ser.*, vol. 1000, no. 1, p. 012049, Apr. 2018, doi: 10.1088/1742-6596/1000/1/012049.

[13] B. Subba and P. Gupta, “A tfidfvectorizer and singular value decomposition based host intrusion detection system framework for detecting anomalous system processes,” *Computers & Security*, vol. 100, p. 102084, Jan. 2021, doi: 10.1016/j.cose.2020.102084.

[14] Y. Shin and K. Kim, “Comparison of Anomaly Detection Accuracy of Host-based Intrusion Detection Systems based on Different Machine Learning Algorithms,” *International Journal of Advanced Computer Science and Applications (IJACSA)*, vol. 11, no. 2, Art. no. 2, 32/29 2020, doi: 10.14569/IJACSA.2020.0110233.

[15] C.-M. Ou, “Host-based Intrusion Detection Systems Inspired by Machine Learning of Agent-Based Artificial Immune Systems,” in *2019 IEEE International Symposium on INnovations in Intelligent SysTems and Applications (INISTA)*, Jul. 2019, pp. 1–5. doi: 10.1109/INISTA.2019.8778269.

[16] Y. ALEKSIEVA, H. VALCHANOV, and V. ALEKSIEVA, “An approach for host based botnet detection system,” in *2019 16th Conference on Electrical Machines, Drives and Power Systems (ELMA)*, Jun. 2019, pp. 1–4. doi: 10.1109/ELMA.2019.8771644.