



Faculty of Engineering and Technology

Master of Computing (MCOM)

Master Thesis

---

## Intrusion Detection System using Feature Ranking and GWO

نظام الكشف عن الاختراقات باستخدام أسلوب تصنيف السمات و  
خوارزمية قطع الذئب الرمادية

---

*Author:*

Ameer Nasrallah

*Supervisor:*

Dr. Iyad Tumar

This Thesis was submitted in partial fulfilment of the requirements  
for the Master's Degree in Computing from the Faculty of Graduate  
Studies at Birzeit University, Palestine.

September 7, 2020



## Intrusion Detection System using Feature Ranking and GWO

By: Ameer Nasrallah

Approved by the thesis committee

---

Dr. Iyad Tumar

---

Dr. Majdi Mafarja

---

Dr. Ahmad Alsadeh

# Abstract

Intrusion detection system (IDS) is an important component in modern network environments. An IDS analyzes network traffic to detect abnormal activities. A challenging task in IDS is the ability to update itself and detect anomalies in short time. Therefore, many machine learning techniques have been applied on IDS. But they still use many attributes of network traffic which sometimes decreases the detection rate and always increases detection time.

In this study, we propose a hybrid feature selection approach to address these challenges. The first stage of the proposed approach is feature ranking using ANOVA F-value. In this stage, a percentage of the top ranked features will be selected. In the second stage, a wrapper approach with GWO as a search strategy, and Random Forest for evaluation is used. In this stage, the reduced dataset from the first stage is the input, and the output is an optimal subset of features.

We tested the proposed approach on NSL-KDD dataset, and compared its performance with the performance of few classifiers without feature selection. We were able to achieve a dimensionality reduction of 68%, while achieving accuracy, detection rate, and false alarm rate similar to those of the chosen classifiers. Moreover, we reduced the search time of GWO by 18%. Moreover, we were able to reduce the training time of KNN and SVM by 77% and 62% respectively in multi-class classification, and by 59% and 68% respectively in binary classification. Similarly, the testing time of KNN and SVM was reduced by 93% and 46% respectively in multi-class classification, and by 89% and 48% respectively in binary classification.

## الملخص

يعد نظام كشف الاختراقات مكوناً مهماً في بيئات الشبكات الحديثة. يقوم هذا النظام بتحليل البيانات التي تمر في الشبكة لاكتشاف الأنشطة غير الاعتيادية. إحدى أصعب المهام في هذه الأنظمة هي القدرة على تحديث نفسها واكتشاف الحالات المريبة في وقت قصير. لذلك، تم تطبيق العديد من تقنيات التعلم الآلي على هذه الأنظمة. وبالرغم من ذلك، لا تزال هذه التقنيات تعتمد على تحليل كل سمات البيانات التي تمر في الشبكة، وهذا بدوره يقلل أحياناً من معدل اكتشاف الهجمات ويزيد دائماً من الوقت المطلوب للاكتشاف.

لمواجهة هذه التحديات، نعرض في هذا البحث طريقة مكونة من خطوتين لاختيار السمات المميزة في بيانات الشبكة. المرحلة الأولى من هذه الطريقة هي تصنيف السمات باستخدام طريقة تعرف باسم تحليل التباين ANOVA F-value. في هذه المرحلة، سيتم اختيار نسبة من أعلى السمات تصنيفاً. هذه البيانات التي تم اختيارها في المرحلة الأولى ستكون مدخلاً للمرحلة الثانية. في المرحلة الثانية، سنقوم باستخدام خوارزمية قطيع الذئب الرمادية GWO كاستراتيجية للبحث عن أفضل السمات.

تم تجربة الطريقة المقترحة على مجموعة بيانات تدعى NSL-KDD، وقارننا أداءها بأداء عدد قليل من المصنفات دون مرحلة اختيار السمات. تمكنا من خفض عدد السمات بنسبة 68%، مع تحقيق تقريبا نفس الدقة ومعدل الكشف ومعدل الإنذار الخاطئ للمصنفات التي قمنا باختيارها. كما و تمكنا من تقليل وقت البحث في خوارزمية GWO بنسبة 18%. بالإضافة إلى ذلك، تمكنا من تقليل وقت تدريب مصنفات KNN و SVM بنسبة 77% و 62% على التوالي في التصنيف متعدد الفئات، و 59% و 68% على التوالي في التصنيف الثنائي. وبالمثل، تم تقليل وقت اختبار مصنفات KNN و SVM بنسبة 93% و 46% على التوالي في التصنيف متعدد الفئات، و 89% و 48% على التوالي في التصنيف الثنائي.

# Acknowledgements

Alhamdulillah, all praises to Allah for the strengths and His blessing in completing this thesis.

It would not have been possible to complete this thesis without the support of the loved ones around me. My dear **mother**, without her support, I would not have even thought of continuing my academic journey. My dear **father**, for him being a role model in determination and hard work. My dear **wife**, for her patience, optimism and support throughout the past two years and throughout the difficult moments. My son **Yahya**, for the joy he brought to our family. I would like to thank my brothers and their families, and my friends.

And of course I would like to thank my supervisor **Dr. Iyad Tumar**, for his patience and his open mind. I have learned a lot from you, not just from academia. And I would like to thank **Dr. Majdi Mafarja** for his kind assistance throughout the journey

# Contents

<b>1</b>	<b>Introduction</b>	<b>10</b>
1.1	Intrusion Detection System (IDS) . . . . .	11
1.2	Machine Learning in Anomaly-based Detection . . . . .	11
1.3	Problem Statement and Motivation . . . . .	13
1.4	Thesis Outline . . . . .	15
<b>2</b>	<b>Foundation and Literature Review</b>	<b>16</b>
2.1	Theoretical Foundation . . . . .	16
2.1.1	Feature Selection . . . . .	19
2.1.1.1	Filter Approaches . . . . .	19
2.1.1.2	Wrapper Approaches . . . . .	20
2.1.1.3	Embedded Approaches . . . . .	20
2.1.1.4	Hybrid Approaches . . . . .	21
2.1.2	Metaheuristic Algorithms . . . . .	21
2.1.2.1	Single-based Metaheuristic Algorithms . . . . .	22
2.1.2.2	Population-based Metaheuristic Algorithms . . . . .	22
2.2	Related Work . . . . .	23
2.3	Summary . . . . .	29
<b>3</b>	<b>Hybrid Feature Selection using Feature Ranking and GWO</b>	<b>31</b>
3.1	Overview . . . . .	31
3.2	NSL-KDD . . . . .	33
3.2.1	Overview . . . . .	33
3.2.2	Description . . . . .	34

3.3	Nominal Features Transformation using Probability Density Function (PDF) . . . . .	38
3.4	Numerical Features Normalization using Min-Max . . . . .	38
3.5	Feature Ranking using ANOVA F-value . . . . .	39
3.6	Feature Selection using BGWO . . . . .	43
3.7	Multi-class Classification using One-vs-rest Strategy . . . . .	47
3.8	Summary . . . . .	50
<b>4</b>	<b>Experimental Setup and Evaluation Metrics</b>	<b>52</b>
4.1	Methodology . . . . .	52
4.2	Environment and Implementation Tools . . . . .	53
4.3	Evaluation Metrics . . . . .	54
<b>5</b>	<b>Result Analysis</b>	<b>59</b>
5.1	Parameters Tuning for GWO . . . . .	59
5.2	Classification Performance and Selected Features . . . . .	66
5.3	Summary . . . . .	85
<b>6</b>	<b>Conclusion and Future Work</b>	<b>86</b>
	<b>References</b>	<b>89</b>
<b>A</b>	<b>Implementation</b>	<b>98</b>

# List of Figures

2.1	Hyperplane example . . . . .	17
2.2	Decision tree example . . . . .	18
2.3	Machine Learning Algorithms in Anomaly-based IDS . . . . .	25
3.1	Proposed approach flowchart . . . . .	32
3.2	F-Distribution $F(4, 26)$ . . . . .	42
3.3	One-vs-rest (OvR) Dataset Separation . . . . .	48
3.4	One-vs-rest (OvR) Approach . . . . .	49
4.1	Example on how to calculate TP, FN and FP for U2R class . . . . .	57
5.1	GWO Convergence curve when Population is 5 . . . . .	60
5.2	GWO Convergence curve when Population is 10 . . . . .	61
5.3	GWO Convergence curve when Population is 20 . . . . .	61
5.4	GWO Selected Features curve when Population is 5 . . . . .	62
5.5	GWO Selected Features curve when Population is 10 . . . . .	62
5.6	GWO Selected Features curve when Population is 20 . . . . .	63
5.7	GWO Avg Training Time when Population is 5 . . . . .	63
5.8	GWO Avg Training Time when Population is 10 . . . . .	64
5.9	GWO Avg Training Time when Population is 20 . . . . .	64
5.10	Average ANOVA F-value for each feature in NSL-KDD. All the 20 features below the horizontal grey line were selected . . . . .	67
5.11	Most frequent features selected by GWO with a threshold of 50% . . .	68
5.12	Classifier Time Performance . . . . .	69
5.13	Classifier Overall Accuracy . . . . .	70
5.14	Normal Class Detection Rate . . . . .	70



5.15 DoS Class Detection Rate . . . . .	71
5.16 Probe Class Detection Rate . . . . .	71
5.17 R2L Class Detection Rate . . . . .	72
5.18 U2R Class Detection Rate . . . . .	72
5.19 Normal Class False Alarm Rate . . . . .	73
5.20 DoS Class False Alarm Rate . . . . .	73
5.21 Probe Class False Alarm Rate . . . . .	74
5.22 R2L Class False Alarm Rate . . . . .	74
5.23 U2R Class False Alarm Rate . . . . .	75
5.24 Normal Class Precision . . . . .	75
5.25 DoS Class Precision . . . . .	76
5.26 Probe Class Precision . . . . .	76
5.27 R2L Class Precision . . . . .	77
5.28 U2R Class Precision . . . . .	77
5.29 Normal Class F1 score . . . . .	78
5.30 DoS Class F1 score . . . . .	78
5.31 Probe Class F1 score . . . . .	79
5.32 R2L Class F1 score . . . . .	79
5.33 U2R Class F1 score . . . . .	80
5.34 Binary Classifier Time Performance . . . . .	81
5.35 Binary Classifier Overall Accuracy . . . . .	82
5.36 Binary Classifier Detection Rate . . . . .	82
5.37 Binary Classifier False Alarm Rate . . . . .	83
5.38 Binary Classifier Precision . . . . .	83
5.39 Binary Classifier F1 score . . . . .	84

# List of Tables

2.1	IDS Datasets created from real network traffic . . . . .	24
3.1	NSL-KDD Attack Distribution . . . . .	34
3.2	NSL-KDD Features . . . . .	36
3.3	NSL-KDD ANOVA Example . . . . .	40
3.4	F-value calculation template . . . . .	42
3.5	F-value calculation for service feature . . . . .	43
3.6	F-value calculation for flag feature . . . . .	43
4.1	Machine specs . . . . .	53
4.2	Confusion matrix . . . . .	55
4.3	Multi-class confusion matrix . . . . .	56
4.4	Normal Confusion matrix . . . . .	58
4.5	DoS Confusion matrix . . . . .	58
4.6	Probe Confusion matrix . . . . .	58
4.7	R2L Confusion matrix . . . . .	58
4.8	U2R Confusion matrix . . . . .	58
5.1	Chosen values for ANOVA F-value threshold, GWO population, and GWO iterations . . . . .	60
5.2	The starting and ending values of the fitness with 10 iterations . . . .	65
5.3	GWO with 50% threshold vs. GWO only . . . . .	67

# Chapter 1

## Introduction

Nowadays, data is being generated, collected, and stored in high volumes and in almost all fields. This high rate of data generation not only requires advanced storage techniques, but it also requires protecting it against unauthorized access. Such protection has become more important, because we are in the age of cloud and social media. These technologies are now collecting sensitive data, including personal and financial data. In fact, it is expected that by 2022, the amount of data being stored by different cloud providers, such as Google, AWS, and Facebook will be increased hundred times [1].

Even though the recent advancements in information and communications technology (ICT) enabled many storage and security technologies to be applicable, they also played a role in increasing the number of network and cyber attacks. A recent statistics on cyber security [2] showed that it is expected to have three trillion cyber attacks by 2021 with high probability of zero-day attacks.

Therefore, and due to the increased number of people that use the internet in daily basis, there is an increased need for suitable protection systems. There have been some security mechanisms, such as data encryption, user authentication, anti-virus, and firewall. But, these traditional mechanisms failed to cope with the variety of attack patterns [3]. Accordingly, intrusion detection system (IDS) became a necessity.

## 1.1 Intrusion Detection System (IDS)

An intrusion detection system (IDS) is a system that automatically checks and analyzes network flow to detect and prevent abnormal activities [4]. This includes monitoring both user and system behaviors, such as the unauthorized access to network resources, and the analysis of network packet fields (e.g. IP address, flag, ports) [5]. Upon detecting an intrusion, the IDS alarms it to the management [6].

Based on the deployment of the IDS, it can be categorized into Host based IDS (HIDS) and Network based IDS (NIDS) [5]. As the name indicates, HIDS is installed on the host computer to check and analyze the log files after an intrusion happens. Therefore, HIDS is not useful for large network environments [4]. On the other hand, NIDS is installed on a network management system. NIDS works in real time to monitor network traffic, and it uses detection algorithms to identify potential intrusions.

Furthermore, an IDS can be classified based on the detection mechanism to knowledge-based [4] and behavior-based [5]. In knowledge-based (or signature-based) detection, the IDS uses misuse detection to detect known attacks by comparing between received packets and a predefined set of collected data (e.g. signature files). While in behavior-based (or anomaly-based) detection, the IDS uses anomaly detection to detect unknown attacks by comparing the system state with the normal activity profile it builds. Behavior-based detection suffers from high false alarm rates. Despite that, it is still considered better than knowledge-based detection as it can detect novel or zero-day attacks. Hence, anomaly-based detection got more attention during the past twenty years.

## 1.2 Machine Learning in Anomaly-based Detection

The main challenge in anomaly-based IDS is to enable it to update itself and detect attacks in short time [4]. Consequently, there have been many research efforts to apply machine learning techniques in intrusion detection. These techniques included few semi-supervised approaches [7], and many supervised approaches [8] [9].

These approaches were applied to common datasets in IDS, such as: NSL-KDD [10], ISCX-IDS-2012 [11] and CIC-IDS-2017 [12].

As other datasets in different fields, IDS datasets usually contain large number of attributes or features. Some of these features are irrelevant and/or redundant. High dimensionality affects the performance of any machine learning technique. And it is called curse of dimensionality. The most common way to address this issue is through dimensionality reduction techniques.

Feature selection is one of the widely used dimensionality reduction techniques. Feature selection is the process of selecting a subset of features according to certain criteria [13]. It is performed as a preprocessing step, and it improves the mining performance.

There are two major aspects to be considered when designing a feature selection method: evaluation and generation (search). From evaluation perspective, filter-based and wrapper-based approaches are used. Filter-based methods evaluate features independently of any learning algorithm, just relying on characteristics of data [14] [15]. Examples of filter-based method include Fisher Score [16], and ANOVA F-value [17]. Wrapper-based methods evaluate features based on a learning algorithm [18]. Examples of wrapper-based method include the LVW algorithm [19] and FVBRM [20]. Due to the lack of a learning algorithm, filter methods usually perform faster than wrapper methods, however, the subset of selected features in filter methods may not represent the optimal subset for classification [15].

From search perspective, the search for an optimal set of features in a high dimensional space is challenging and most of the time it is not practical. Metaheuristic algorithms are used as a solution to this problem. A metaheuristic algorithm is an optimization problem suitable for real-world problems, because it searches for an optimal or near-optimal solution by making some reasonable assumptions on the problem. Some metaheuristic algorithms were used in IDS, such as the usage of random mutation hill climbing in [18] and the usage of tabu search in [21].

Many metaheuristic algorithms take a lot of time to converge towards a solution and they may stuck at a local optima. As a consequence, most researchers prefer working with two categories of metaheuristic algorithms: evolutionary computation (EC) algorithms and swarm intelligence (SI). Evolutionary computation (EC) algo-

rithms are based on biological evolution and natural selection techniques. Genetic algorithm (GA) [22] is a common EC algorithm. Swarm intelligence algorithms, on the other hand, are inspired from nature phenomena and social behavior of animals, birds, wolves, etc [23]. Particle Swarm Optimization (PSO) [24], Ant Colony Optimization (ACO) [25], Firefly Algorithm (FA) [26] are common SI algorithms. More recently, a promising SI algorithm emerged. This algorithm is call Grey wolf optimization (GWO) [27].

Grey wolf optimization (GWO) is a swarm intelligence algorithm that imitates the hunting process of a pack of grey wolves in nature [28]. It has shown faster conversion than PSO along with a good balance between exploration and exploitation. This means that it well manages the trade-off between local optima and global optima. To apply GWO in feature selection, Emary et al. [23] developed a binary version that has been used in several IDSs [29] [30].

### 1.3 Problem Statement and Motivation

The first challenge in intrusion detection systems is to reduce the number of features and increase the detection accuracy. We have reviewed in the previous section feature selection as a solution to this problem. The second challenge is to have short training time, and a real-time or near real-time prediction.

Although there have been many studies on IDS to address the first challenge, few of them considered the second challenge. So, there are two main challenges to address in this study:

1. Reduce the data dimensionality to have short training time, and a real-time or near real-time detection.
2. Achieve high detection rate, high accuracy, and low false alarm rate at the same time.

In this study, we propose a hybrid feature selection approach to address these challenges. A hybrid approach is a two-stage feature selection approach. In the first stage, a filter method is used to reduce the data dimensionality. The reduced data is the input to the second stage. In the second stage, a wrapper method is used

to select the optimal feature subset. The benefit of this approach is that it offers the fast computation of filter methods and the optimization capabilities of wrapper methods.

In the first stage of the proposed approach, we will use a filter method called Analysis of Variance (ANOVA) F-value, which is a well-known feature ranking method. In the second stage, we will use a wrapper method that uses binary GWO (BGWO) as a search strategy, and a Random Forest classifier for evaluation.

To assist on the performance of the proposed approach, we will consider NSL-KDD dataset. In the future work, we will consider larger datasets, such as CIC-IDS-2017 and CSE-CIC-IDS-2018. Moreover, we will compare the performance of our approach against the performance without the proposed feature selection, and the performance when GWO is used alone. We will use five common classifiers in IDS literature: Decision Tree, SVM, KNN, ANN, and Naive Bayes. And finally, we will consider both multi-class classification (using one-vs-rest strategy) and binary classification.

Throughout the experiments, we have noticed that Decision Tree and ANN classifiers are already performing well without feature selection. While, SVM and KNN are taking long time to do the training and testing. On the other hand, Naive Bayes is the worst performing classifier.

Using the proposed approach, we have been able to achieve the following:

- Dimensionality reduction of **68%** from 41 to 13 features.
- Reduce GWO search time by **18%**.
- Improve Naive Bayes overall accuracy in multi-class classification.
- Reduce the training time of KNN by **77%** in multi-class classification, and by **59%** in binary classification.
- Reduce the testing time of KNN by **93%** in multi-class classification, and by **89%** in binary classification.
- Reduce the training time of SVM by **62%** in multi-class classification, and by **68%** in binary classification.

- Reduce the testing time of SVM by **46%** in multi-class classification, and by **48%** in binary classification.
- Almost preserve the overall accuracy, detection rates and false alarm rates of the other classifiers. In fact, our numbers were not better than the classifiers without feature selection, but at least they were around with less time and less features.

## 1.4 Thesis Outline

- In Chapter 2, we will present a theoretical background on machine learning, feature selection and metaheuristic algorithms. Then, we will discuss few studies that used different machine learning approaches on IDS.
- In Chapter 3, we will discuss each step of our proposed approach along with mathematical foundations, and explanatory examples.
- In Chapter 4, we will describe the methodology, environment, and tools we used to setup and implement the experiments. Then, we will illustrate the evaluation metrics used in multi-class classification and binary classification.
- In Chapter 5, we will present and analyze experiments results, along with justification on chosen parameters for GWO and ANOVA F-value.
- And finally, we will present conclusion and future work in Chapter 6.



# Chapter 2

## Foundation and Literature Review

In this chapter, we first present a theoretical background on machine learning, feature selection and metaheuristic algorithms. Then, we present the related work in intrusion detection systems.

### 2.1 Theoretical Foundation

Machine learning is a special branch of artificial intelligence that makes a decision or predicts an output by acquiring knowledge from existing input data. Arthur Samuel defined machine learning in 1959 as a study that allows computers to learn knowledge without being programmed. There are three main categories for machine learning techniques:

1. Supervised Learning: also called Classification. In supervised learning, the input training data is already labeled, or in other terms already classified. Some of the most popular supervised learning algorithms are: Bayesian Networks, Decision Trees, Artificial Neural Networks, K Nearest Neighbor, and Support Vector Machine. We will come to each of those in short.
2. Unsupervised Learning: also called Clustering. Unlike supervised learning, input training data here is unlabeled. Some of the most popular unsupervised learning algorithms are: K-means clustering, Apriori algorithm, and Fuzzy clustering.
3. Semi-supervised Learning: this category makes use of labeled and unlabeled

data. Basically, labeled samples will be used to learn the classes, and unlabeled samples will be used to find the suitable separation between classes. Graph-based and heuristic-based algorithms are used in this category.

In this study, we will use the following supervised learning algorithms (classifiers):

#### A. Support Vector Machine (SVM)

A Support Vector Machine (SVM) [31] is a supervised machine learning algorithm used for classification and regression purposes. The basic idea in SVM is to find a hyperplane that best divides a dataset into multiple classes.

A hyperplane can be a line as in Figure 2.1 [32], but this is considered a simple example with only two dimensions that do not overlap. In real applications, SVM is used with any number of dimensions, and therefore, a hyperplane is considered a generalization of a plane where it can be a point in one dimension, a line in two dimensions and a plane in three dimensions [33] [34].

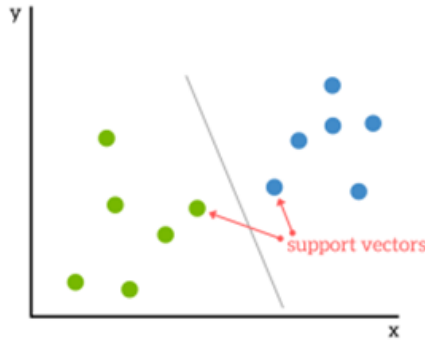


Figure 2.1: Hyperplane example

#### B. K-Nearest Neighbor (KNN)

K-nearest neighbor (KNN) [35] is a supervised learning algorithm commonly used for classification. When classifying an instance sample, KNN computes the minimum distance from that instance to the training sample. Then, the instance sample is classified based on the majority of the K-nearest neighbor category [23].

#### C. Decision Tree

A decision tree is often represented as a flowchart tree where each internal node represents an attribute, and each branch represents a value or range of

this attribute and each leaf node represents a class label. This is depicted in Figure 2.2 [36].

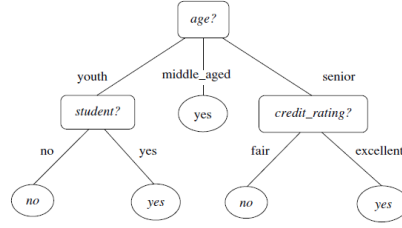


Figure 2.2: Decision tree example

#### D. Naive Bayes

Naive Bayes is based on Bayes' theorem of posterior probability with the assumption of class-conditional independence [36]. This means that it assumes that the impact of an attribute on a class is not affected by the values of other attributes. Therefore, Naive Bayes is usually used because of its simplicity and fast computation [37]. However, this may come at the cost of having low accuracy.

#### E. Artificial Neural Network (ANN)

The idea of Artificial Neural Network (ANN) came from the human brain [37]. Each node (or neuron) in the network is capable of perception, pattern recognition and any other function. The interconnections between the nodes have weights. When a pattern is represented to the input layer of the neural network, it passes this pattern based on the weights to another hidden layer for processing. There can be as many as needed hidden layers. At the end, the last hidden layer passes its processing results to the output layer, which takes the classification decision.

This was a general overview on machine learning, its techniques and some of the well-known supervised algorithms. In the next subsection, we discuss feature selection which is an important topic when classifying large datasets. In the last subsection, we discuss metaheuristic algorithms showing their vital role in feature selection.

### 2.1.1 Feature Selection

Nowadays, data has become essential in every field, and it is increasing dramatically on daily basis. Intrusion detection is no exception. In fact, the recent efforts made by the Canadian Institute for Cybersecurity<sup>1</sup> emphasized the importance of having large datasets to simulate real-life network traffic. Classifying such volumes of data accurately and near real-time is a challenging task for machine learning algorithms. Additionally, such data usually have many attributes to analyze. Not only this causes computation time and space complexities, but it also cause overfitting problems. This is known in the literature as curse of dimensionality [15].

The most well-known way to address this challenge is to use dimensionality reduction techniques. These techniques are mainly categorized into: feature extraction and feature selection. In feature extraction, a low dimensional feature space is constructed either linearly or non-linearly from the original feature space. Principle Component Analysis (PCA) [38] is the most popular feature extraction method. There are other methods as well, such as: Independent Component Analysis (ICA) [39], Linear Discriminant Analysis (LDA) [40], and Locally Linear Embedding (LLE) [41]. In contrast, feature selection removes redundant and irrelevant features ending up with the subset of relevant features. In real world problems, feature selection is more preferred than feature extraction because it offers better readability and interpretability [15].

Feature selection approaches are mainly categorized as: filter approaches, wrapper approaches, embedded approaches, and hybrid approaches.

#### 2.1.1.1 Filter Approaches

Filter feature selection methods evaluate features without utilizing any classification algorithms, just relying on the characteristics of the data [42]. Typically, filter algorithms go through two steps: feature ranking based on certain criteria (e.g. using statistical-based techniques), and selecting the highest ranked features based on a certain threshold. In the first step, features are evaluated on either a univariate or a multivariate schemes [15]. In the univariate scheme, each feature is ranked independently of the feature space, while the multivariate scheme evaluates features

---

<sup>1</sup><https://www.unb.ca/cic/>

in a batch way. Some of the common filter methods include: Fisher Score [16], ReliefF [43], ANOVA F-value [17], Chi-square [44], and Gini Index [45].

Apparently, filter methods are computationally fast. However, they do not guarantee selecting the optimal or near-optimal subset of features that maximizes classification performance.

#### **2.1.1.2 Wrapper Approaches**

In contrary to filter methods, wrapper methods depend on a learning algorithm to evaluate the quality of the selected features [15]. There are two main concepts in any wrapper algorithm: search strategy and evaluation criteria. To clarify, any wrapper algorithm works iteratively where it first searches for a subset of features, then it evaluates the performance of this subset to check if it meets the needed quality. If not, the iteration continues until either finding the optimal subset or reaching a maximum iteration. Search strategies in typical wrapper methods often follows either a recursive feature elimination scheme or a sequential feature selection scheme. Common wrapper methods in IDS are: Feature Vitality Based Reduction Method (FVBRM) [20], Sequential Forward Selection (SFS), Best First Search (BFS) and Consistency Subset Eval (CSE).

Even though wrapper methods achieve better classification performance than filter method, they still suffer from high computation, overfitting and large search spaces [42].

#### **2.1.1.3 Embedded Approaches**

On the one hand, filter methods do not incorporate learning. On the other hand, wrapper methods suffer from high computation and large search spaces. Therefore, Embedded methods offer a trade-off solution between filter methods and wrapper methods [15]. Embedded methods are similar to wrapper methods, except that they consider feature selection within the learning algorithm. In other words, these methods do not search iteratively for the best subset. Rather, they evaluate the importance of each feature to the prediction process. Most widely used embedded methods are regularization-based, such as LASSO regression [46], and tree-based such as decision trees.

#### 2.1.1.4 Hybrid Approaches

Hybrid approaches are two-stage feature selection methods [42]. In the first stage, a filter method is applied to select a subset of features. In the second stage, a wrapper method is used to select the optimal subset of features from the first subset. The main benefit of these approaches is that they take advantage of the fast computation of filter methods to reduce the search space of wrapper methods. Consequently, wrapper methods become faster and they do not overfit. In this study, we are adopting this approach as will be explained in Chapter 3.

### 2.1.2 Metaheuristic Algorithms

Selecting the subset of features that maximizes classification performance can be seen as an optimization problem. But since we are dealing with this problem in intrusion detection context, which is a real-world context, there are few challenges to take into consideration. First, it is usual to have insufficient or imperfect information. Second, it can be computationally expensive to reach the exact optimal solution.

Taking these challenges into consideration, it is recommended to use what is called metaheuristic algorithms for such problems [42]. In simple terms, heuristic means to find or discover by *trial-and-error*. This means that metaheuristic is a higher-level heuristic that guides other heuristics to reach solutions beyond the local optimum [47].

Compared to other optimization algorithms, metaheuristic algorithms are suitable for real-world problems, because they tend to simplify calculations by making reasonable assumptions on the optimization problem [42]. The idea is to find a mathematical model that represents the problem. Then, the search for the best solution starts by proposing an initial solution out of the mathematical model. After that, the search process is guided by the information collected during the search itself. This continues until a near-optimal solution is reached.

There are two important components that any metaheuristic algorithm must take care of [48]:

- Exploration: it is called also diversification. It means to search for different

solutions in the search space at a global scale.

- **Exploitation:** it is called also intensification. It means that a good solution is found in the current local region, so the search should be focused in this region.

A good metaheuristic algorithm tries to balance between exploration and exploitation so that it converges fast to the optimal solution. That is why most metaheuristic algorithms are stochastic in nature, which means they use some randomness to make a trade-off between global optima and local optima [49].

Metaheuristic algorithms can be categorized into: single-based metaheuristic and population-based metaheuristic [42]. In the following subsections, we briefly discuss these categories, and we present some of the common metaheuristic algorithms used in feature selection.

#### **2.1.2.1 Single-based Metaheuristic Algorithms**

These algorithms are also called trajectory algorithms. The reason behind this is that they create an initial solution, and then they follow a specific path in the search process to investigate neighboring regions [42]. The aim of these algorithms is to reach a local optima. Simulated Annealing (SA) [50] and Tabu Search (TS) [51] are two common examples of this algorithm.

#### **2.1.2.2 Population-based Metaheuristic Algorithms**

Compared to single-based algorithms, these algorithms initialize a set of solutions (population), and then they search for better population. This continues until a search criteria is reached [42]. Most of the algorithms in this category fall into either Evolutionary Computation (EC) algorithms or Swarm Intelligence (SI) algorithms.

Evolutionary Computation (EC) is a form of Computation Intelligence (CI) that is inspired by biological evolution and natural selection techniques [52]. These algorithms start by a randomly generated population. Then, evolution operations such as mutation are used to evolve the population. Genetic Algorithm (GA) [22] and Differential Evolution (DE) [53] are two common evolutionary algorithms.

Swarm Intelligence (SI) is also a form of CI which simulates and imitates the natural swarms or communities, such as bird swarms and fish schools. These algorithms

focus on finding mathematical formulas to model the behavior of swarm's members and their interactions to find food sources [3]. There are many SI algorithms, such as: Particle Swarm Optimization (PSO) [24], Firefly Algorithm (FA) [26], Ant Colony Optimization (ACO) [25], Artificial Bee Colony (ABC) [54], and Grey Wolf Optimizer (GWO) [27].

In this study, we are mainly adopting Grey Wolf Optimizer (GWO) algorithm which will serve as the search strategy of the hybrid approach proposed in Chapter 3. GWO is a swarm intelligence algorithm developed by Mirjalili et al. [27]. It mimics the hunting process of a pack of grey wolves in nature. A binary version of GWO was developed by Emary et al. [23] to find optimal regions of complex search spaces which makes it useful for feature selection. The binary version of GWO showed faster convergence speed and higher classification accuracy than other population-based algorithms, such as GA and PSO. Furthermore, GWO achieves a good balance between exploration and exploitation as will be shown mathematically in Section 3.6.

## 2.2 Related Work

Many network attacks and abnormal behaviors have been introduced in recent years. As a result, traditional ways for intrusion detection were not able to cope with the speed of network advancements. For example, deep packet inspection (DPI) is considered a traditional way that uses a rule-based technique for intrusion detection. A DPI solution usually captures and decapsulates all packets passing through the network. Then, it applies certain rules stored in the database to the packet in order to identify anomalies. Applying such rules fails to understand several attack behaviors in modern network environments [4].

As a consequence, researchers in the last two decades considered intrusion detection as a classification problem [3]. Since then, there have been an ongoing effort to create datasets simulated from real network traffic to capture the evolving attacks. Table 2.1 lists most common IDS datasets [1]. In the scope of this study, we consider only NSL-KDD dataset (analyzed in Section 3.2), and in the future we will be considering more recent and more realistic datasets such as CIC-IDS-2017 and CSE-CIC-IDS-2018.



Table 2.1: IDS Datasets created from real network traffic

Dataset Name	Created by	Year	Number of Features
DARPA [55]	MIT Lincoln Laboratory	1998	41
KDDCUP'99 [56]	University of California	1999	41
NSL-KDD [10]	University of California	2009	41
Kyoto [57]	Kyoto University	2011	24
ISCX-IDS-2012 [11]	University of New Brunswick	2012	IP flows
CIC-IDS-2017 [12]	Canadian Institute of Cyber Security	2017	80
CSE-CIC-IDS-2018 [12]	Canadian Institute of Cyber Security	2018	80

Alongside this advancement in IDS datasets, there have been many studies applying different machine learning approaches to IDS problems. Figure 2.3 presents most of these approaches according to latest surveys [58] [59] [60] [42].

The main motivation to use machine learning and data mining techniques is that they detect known and unknown attacks in the network. One way to detect unknown attacks is throw unsupervised and semi-supervised learning algorithms [7] [61] [62] [63]. Nonetheless, supervised learning algorithms are still more dominant in the field of intrusion detection. This is mainly because they are easier to compare and to build on top of them.

ANN and SVM are the most used supervised learning algorithms for intrusion detection [37] [59]. Saied et al. [64] used ANN to detect unknown distributed denial-of-service (DDoS) attack, and they were able to detect 100% of the known attacks and 95% of the unknown attacks. Yin et al. [65] used also recurrent neural networks (RNN) to detect attacks. They achieved about 99% detection rate for known attacks and more than 68% for the unknown attacks using NSL-KDD dataset. Another study [66] proposed a hybrid of multilayer perceptron (MLP) neural network and radial basis function (RBF) neural network to improve prediction accuracy in IDS. Even though these classifiers perform well, they still suffer computationally from high dimensional data. Therefore, Yi et al. [67] suggested a new kernel function U-RBF to

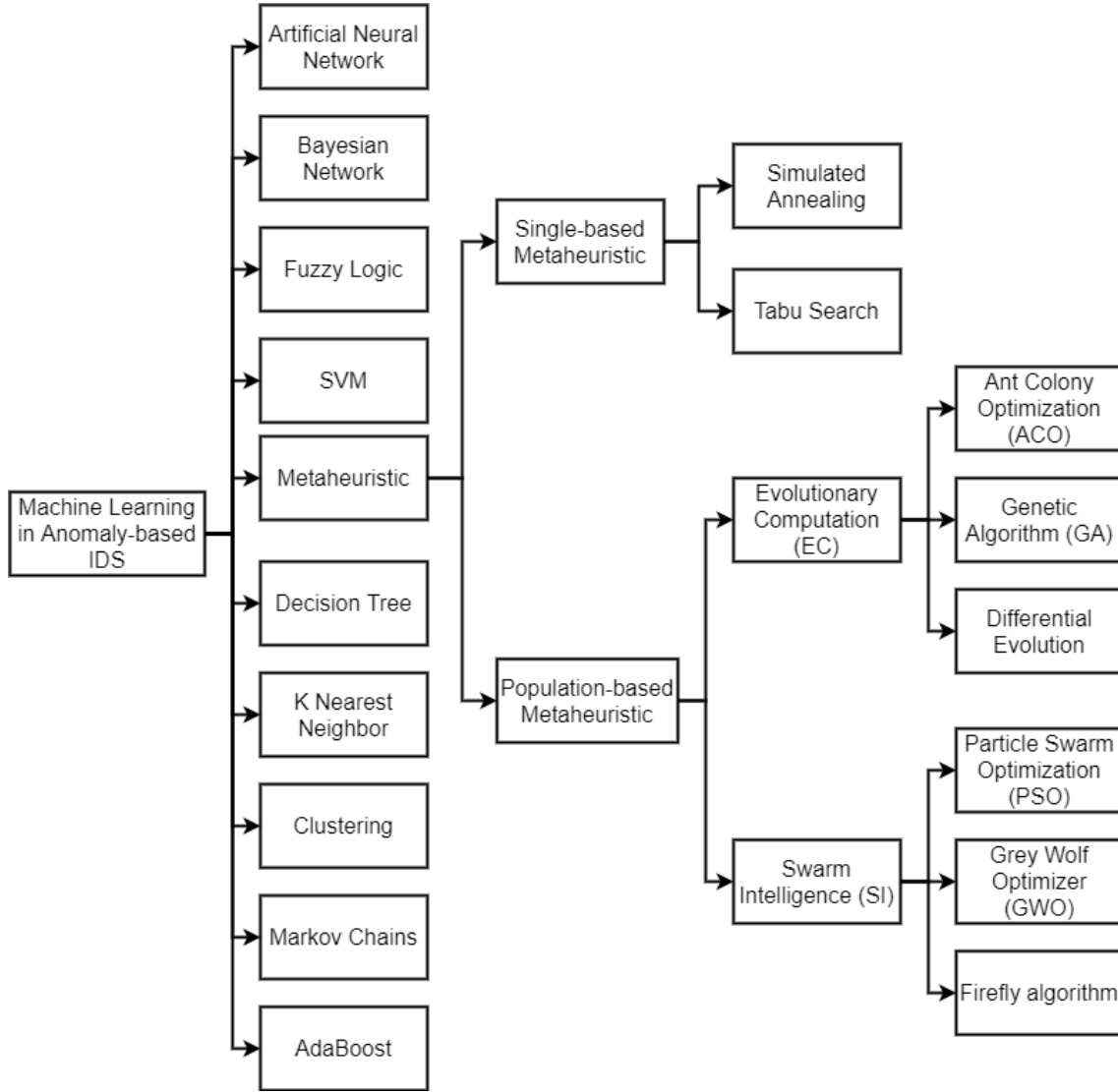


Figure 2.3: Machine Learning Algorithms in Anomaly-based IDS

SVM to reduce its computation complexity while achieving similar performance on KDDCUP'99 dataset. Koc et al. [68] proposed another idea which is to use Bayesian network classifiers as they are known to be simpler and faster. They showed that Hidden Naïve Bayes (HNB) performed better than SVM in detecting DoS attacks.

Nevertheless, many researchers pointed out that dimensionality reduction is vital to achieve good performance in terms of high accuracy, low false positive rate and efficient computation. For example, Vasan and Surendiran [8] did experiments on KDDCUP'99 and UNB ISCX datasets using Principal Component Analysis (PCA), and obtained 10 principal components to achieve similar accuracy to 41 features used by C4.5 classifier.

Among dimensionality reduction techniques, feature selection is the most popular

technique. A recent systematic review [60] showed that feature selection has been broadly adopted in the last decade. Moreover, wrapper-based methods and hybrid methods are being used more than filter-based methods. This is because wrapper-based methods and hybrid methods search for the optimal or near-optimal subset of features. Since we are looking for optimal solutions, most wrapper-methods use metaheuristic algorithms as a search strategy technique.

Metaheuristic algorithms aim to find the optimal or near-optimal solution to a problem. As explained in Section 2.1, these algorithms are divided into: single-based and population-based. In literature [42], population-based metaheuristic algorithms are more commonly used for intrusion detection compared to single-based metaheuristic algorithms. Among population-based methods, Evolutionary Computation (EC) and Swarm Intelligence are the most common methods.

As an example on single-based metaheuristic, Li et al. [18] proposed a modified version of random mutation hill climbing (RMHL) as a search strategy and Linear SVM for evaluation to find the optimal feature subset per each attack category of KDDCUP'99 dataset. The modified version of RMHL speeds up its convergence and its dimensionality reduction ability. They performed the experiments on five samples each representing an attack category of KDDCUP'99.

To assist on the effectiveness of feature selection, the authors chose to compare building and testing time, and ROC score of selected features against all features. The results showed that the selected features improves the speed of the algorithm while achieving higher ROC score.

When working with population-based metaheuristic algorithms, the researchers focus on effectively reducing the search space of such algorithms. This is usually achieved by applying a filter-based method in prior of wrapper-based methods. For example, Mohammadi et al. [69] focused on the importance of feature selection to reduce false positive rate and to reduce the search space for large datasets such as KDDCUP'99. First, they applied a filter-based method called feature grouping based on linear correlation coefficient (FGLCC) to select the best features that have maximum correlation with the class and minimum relation with other selected features. Then, they applied a wrapper-based method that uses cuttlefish algorithm (CFA) as a search strategy and decision tree classifier for the fitness function.

The proposed approach only considered binary classification, and its performance was evaluated on 10% of KDDCUP'99 dataset. The results showed good accuracy and low false positive rate.

Similarly, Selvakumar and Muneeswaran [5] used filter and wrapper methods to reduce the dimensionality of KDDCUP'99 dataset. First, they used the mutual information as a filter method to select the first subset of features. Then, they used a metaheuristic technique called Mutual Information Firefly Algorithm (MIFA) to select two subsets of features, the first subset is selected using C4.5 classifier, and the second subset is selected using Bayesian network. The final subset of features is constructed using a voting technique, where each feature is selected if and only if it exists at minimum in two subsets. Finally, C4.5 and Bayesian network are used for classification after feature selection.

They [5] ran the experiments using 10 fireflies and a maximum of 100 iteration. They were able to prove that 10 features (out of 41) are sufficient to improve the training and testing time, the detection rate and the false positive rate.

Regular metaheuristic algorithms suffer from the computation time needed to choose optimal subset and from local optima problem. This is where Evolutionary Computation (EC) and Swarm Intelligence (SI) algorithms come into play. EC and SI algorithms offer similar benefits except that SI algorithms are nature-inspired and tend to converge faster. Now, we will go through some of the recent studies applying EC or SI algorithms in IDS.

Hajimirzaei and Navimipour [70] classified IDS in cloud computing as an NP-Hard problem. Therefore, they suggested to use metaheuristic algorithms and evolutionary methods to solve it. They first used Fuzzy C-Means clustering (FCM) to improve the training speed. Then, they used a multilayer ANN with back propagation for the classification. Moreover, they applied ABC to optimize the linkage weights and biases of the nodes in the training stage.

The experiments were performed on NSL-KDD dataset. The proposed method outperformed similar state-of-the-art methods by achieving lower mean absolute error (MAE) and root mean square error (RMSE) values.

On the other hand, Aburomman and Reaz [71] achieved an accuracy of 92% on KDDCUP'99 dataset by applying particle swarm optimizer (PSO) to the output of

an ensemble of classifiers.

Moreover, Bamakan et al. [3] adopted an enhanced version of PSO called time-varying chaos PSO (TVCPSTO) to select the most important features, and to fine tune the parameters SVM and Multiple Criteria Linear Programming (MCLP) classifiers. The authors emphasized that three main factors impact the efficiency of any intrusion detection system: feature selection, high detection rate, and low false alarm rate. Therefore, they proposed a weighted objective function that takes these three factors into consideration.

NSL-KDD dataset was considered in the experiments. They ran the experiments for 10 iterations, in each iteration 10-fold cross-validation was used, and then the average measures were taken. They proved that feature selection increases the detection rate of their proposed methods. Moreover, they achieved comparable accuracy and false alarm rate to similar methodologies.

More recently, Hajisalem and Babaie [6] used Swarm Intelligence techniques to improve the classification accuracy on NSL-KDD and UNSW-NB15 datasets. To decrease complexity and increase efficiency, they first used Fuzzy C-Means clustering (FCM) to divide the training data into smaller subsets. Then, they applied Correlation-based Feature Selection (CFS) on each subset, which resulted in selecting six features from NSL-KDD and six features in UNSW-NB15. Then, they used Classification and Regression Tree (CART) to generate effective rules set. For example, the authors found that when *logged\_in* feature has the value '0', this demonstrates a normal behavior. During this step, the authors used also the idea of dividing normal behaviors into  $K$  ranges which was already proposed in [72] to reduce processing time. Finally, they applied a hybrid classifier based on Artificial Bee Colony (ABC) and Artificial Fish Swarm (AFS).

The experiments were performed on random samples from NSL-KDD and UNSW-NB15 datasets. The authors [6] were able to achieve an accuracy ranging from 96.7% to 99% and FPR ranging from 0.82% to 0.01% in NSL-KDD, and an accuracy ranging from 95% to 98.9% and FPR ranging from 2.1% to 0.13% in UNSW-NB15.

Even though GWO is still a new SI algorithm, it is taking more attention in IDS recent studies. For example, Seth and Chandra [29] used GWO to reduce the number of attributes in NSL-KDD. They started by applying a probability density

function on nominal attributes to transform them into numerical attributes. Then, they applied a binary version of GWO [23] with a population size of 12 to reduce the NSL-KDD attributes. They were able to select 24 attributes out of 41. Finally, they used neural network as a classifier.

The experiments were performed with different number of neurons, and a size of 70%, 15%, and 15% for the training, testing and validation respectively. Throughout 300 iterations, a maximum accuracy of 99.5% was achieved.

Going further, Devi and Suganthe [30] applied a multi-objective GWO technique for attribute reduction. This technique runs in two stages to combine the efficiency of filter-based methods and the effectiveness of wrapper-based methods. The first stage searches for the attribute combination that maximizes the mutual information. The solution resulted from the first stage is used as an initial population to the second stage. In the second stage, the feature set that maximizes the correct classification ratio (CCR) is selected. A combination of SVM and Naive Bayes classifiers is used to improve the accuracy. Moreover, the factor that controls the exploitation and exploration of GWO is kept in this stage between 0 and 1 to fine tune the solutions around the initial population.

The proposed technique was experimented on NSL-KDD dataset, where it reduced the features from 41 to 18, and achieved good accuracy and false positive rates in a reasonable execution time.

## 2.3 Summary

In this chapter, we first presented a theoretical background on machine learning, feature selection, and metaheuristic algorithms. We also talked briefly about five common classifiers: SVM, KNN, ANN, Decision Tree, and Naive Bayes.

Regarding feature selection, there are four known approaches: filter approaches, wrapper approaches, embedded approaches, and hybrid approaches. Filter approaches are known for their fast computation, while wrapper approaches are known for their optimization capabilities. Embedded approaches are similar to wrapper approaches except that they try to employ feature selection in the learning process, and thus, they are faster. On the other hand, hybrid approaches are two-stage fea-

ture selection approaches, in which the first stage uses a filter method to reduce the data, and the second stage uses a wrapper method that takes the reduced data as input.

Metaheuristic algorithms are optimization algorithms that can be used to search for the optimal or near-optimal subset of features. There are two main categories of metaheuristic algorithms: single-based or trajectory-based algorithms, and population-based algorithms. The main differences between single-based algorithms and population-based algorithms are:

1. Single-based algorithms use one solution while population-based algorithms use a set of solutions.
2. Single-based algorithms look for the local optima, while population-based algorithms make a trade-off between local optima and global optima.

There are two major categories of population-based algorithms: Evolutionary Computation (EC) algorithms and Swarm Intelligence algorithms (SI).

After this theoretical foundation, we reviewed different machine learning algorithms used in IDS. It is noticed that in the last decade, there is an increase in the research studies that consider SI algorithms in their intrusion detection systems. GWO is among these SI algorithms.

# Chapter 3

## Hybrid Feature Selection using Feature Ranking and GWO

As stated in Section 2.1.1, high dimensionality is a challenge that any IDS must overcome to achieve real-time prediction. Therefore, many researchers emphasized the importance and effectiveness of using a dimensionality reduction technique in IDSs [3] [73] [74].

Consequently, we decided to use one of the most efficient dimensionality reduction techniques, which is feature selection. Feature selection is performed in the pre-processing step. Its main objective is to select the features set that will not only decrease classifier training time and prediction time, but will also achieve similar or even better prediction.

In this chapter, we present a hybrid feature selection approach applied on NSL-KDD dataset. The first stage of the feature selection is filter-based using Analysis of Variance (ANOVA) F-value for feature ranking, and the second stage is wrapper-based using Binary Grey Wolf Optimizer (BGWO) as a search strategy and Random Forest for fitness function.

We first present an overview of the approach. Then, we present an overview on NSL-KDD dataset. Then, the next sections go through each step of the approach.

### 3.1 Overview

As shown in Figure 3.1, there are three main steps:



1. Splitting dataset into training and testing sets: a random seed is used to shuffle the data and split it into 70% training and 30% testing sets. Stratified sampling is use in this step, such that the distribution of each class in the complete dataset is preserved in the training and testing sets.

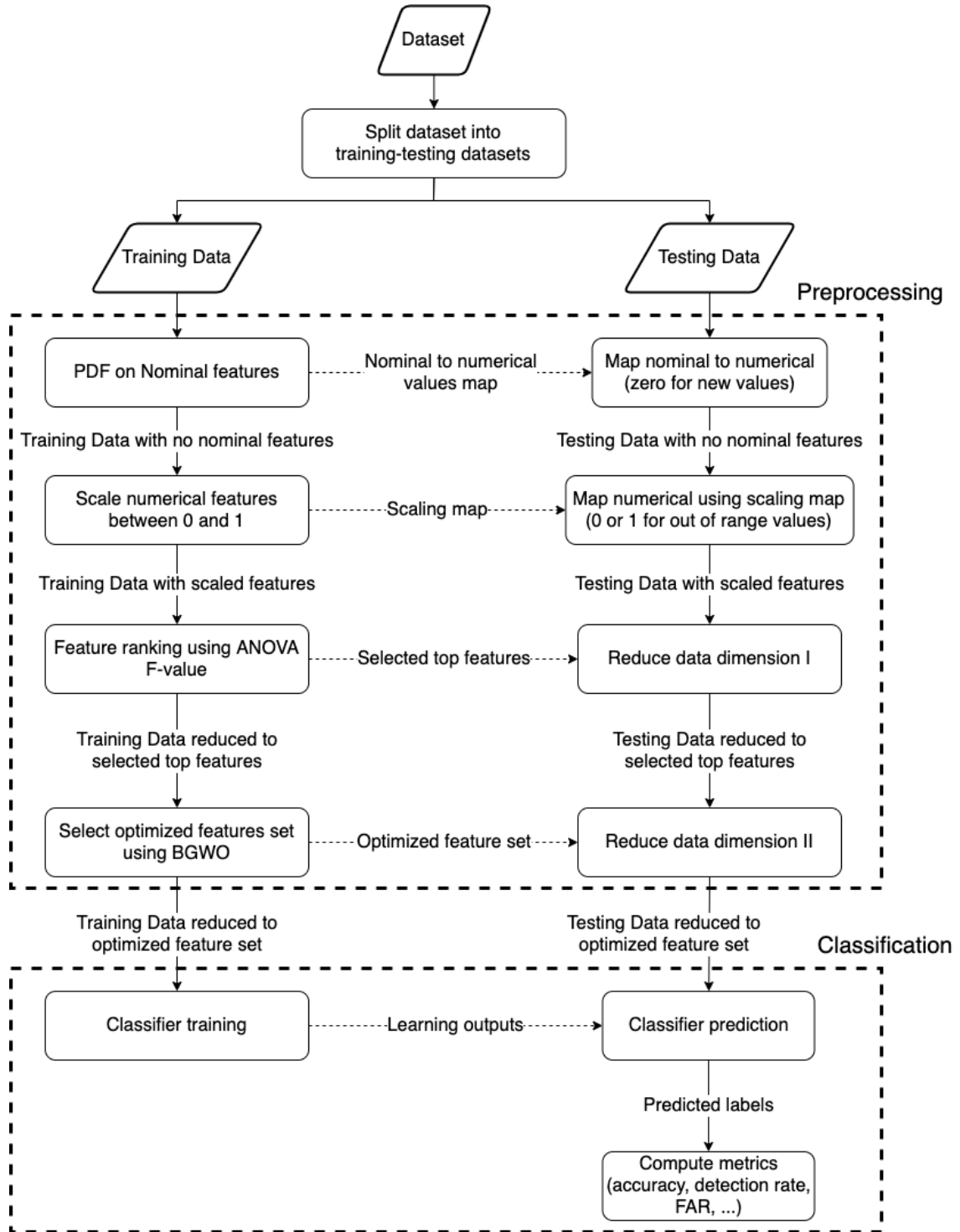


Figure 3.1: Proposed approach flowchart

2. Preprocessing: this is the main step in the proposed approach. Feature ranking (a filter-based method) is used to reduce the dataset by selecting a percentage of the top ranked features. Then, a wrapper-based method with BGWO as a search strategy is applied on the reduced dataset to select the optimal features subset.
3. Classification: different classifiers are applied on the reduced dataset. And both multi-class classification and binary classification are evaluated.

## 3.2 NSL-KDD

### 3.2.1 Overview

NSL-KDD dataset [10] is one of the most effective datasets in the domain of intrusion detection. It is a modified version of KDDCUP'99 dataset [56], which was created in 1999. KDDCUP'99 is constructed from simulated TCP connections in a military network environment [3]. KDDCUP'99 had been the most widely used dataset to evaluate IDSs until recent years [60]. However, researchers found some deficiencies that make it less reliable [10]:

1. Redundant records: this mainly affects the performance of any classifier such that it is biased towards more frequent records.
2. Low difficulty level: applying simple machine learning methods will give at least 86% accuracy, which makes it difficult to compare the different models as they will fall in the range of 86% to 100%.

To deal with these deficiencies, the following improvements were applied to NSL-KDD [10]:

1. Removing all redundant records from train and test sets so that there will be no biasing.
2. Better sampling and distribution for the records which will increase the classification challenge.
3. Reasonable number of records in train and test sets. This makes it affordable to run experiments on the whole dataset without any need for sampling.

NSL-KDD still does not perfectly represent real networks. Nonetheless, it is still a reliable benchmark dataset to compare intrusion detection methods.

### 3.2.2 Description

NSL-KDD records are labelled as normal or attack. There are 39 different attacks distributed (with some overlap) as 22 attacks in the training set and 37 attacks in the testing set. These attacks fall into four basic categories detailed as follows:

- Denial of Service Attack (DoS): involves attacks which try to keep the machine's memory or computing resources too busy such that the machine cannot serve its legitimate users.
- User to Root Attack (U2R): involves attacks in which the attacker first gains access to a normal user account, and then tries to exploit some vulnerability to gain root access to the system.
- Remote to Local Attack (R2L): involves attacks in which attacker keeps sending packets to a machine over some network. The main purpose in these attacks is to try to find a system vulnerability to gain access as a normal user.
- Probing Attack: these attacks scan the computer networks to find some vulnerability in its security controls.

Table 3.1 shows the detailed distribution of the different attacks.

Table 3.1: NSL-KDD Attack Distribution

Attack Category	Attacks Included
DoS	neptune, back, land, pod, smurf, teardrop, mailbomb, apache2, processtable, udpstorm, worm
Probing	ipsweep, nmap, portsweep, satan, mscan, saint
R2L	ftp_write, guess_passwd, imap, multihop, phf, spy, warez-client, warezmaster, sendmail, named, snmpgetattack, snmpguess, xlock, xsnoop, httptunnel
U2R	buffer_overflow, loadmodule, perl, rootkit, ps, sqlattack, xterm

Moreover, NSL-KDD is constructed from 41 attributes or features. These features fall into three main categories as shown in Table 3.2:

1. Basic features: these attributes are extracted from a TCP/IP connection.
2. Content features: these attributes are extracted from the data portion of the packet. They are very important to detect R2L and U2R attacks. This is because these attacks usually involve a single connection.
3. Traffic features:
  - (a) Time-based traffic features: these attributes are extracted from connections in the past two seconds that have the same destination or same service as current connection.
  - (b) Connection-based traffic features: these attributes are extracted from last 100 connections that has the same destination or same service as current connection. Extracting such attributes contributes more in detecting probing attacks.

Table 3.2: NSL-KDD Features

Type	Features	Range/Values
Basic features	duration	0 - 57715
	protocol_type	tcp, udp, icmp
	service	http, private, domain_u, smtp, ftp_data, other, eco_i, telnet, ecr_i, ftp, finger, pop_3, auth, imap4, Z39_50, uucp, courier, bgp, iso_tsap, uucp_path, whois, time, nnsf, vmnet, urp_i, domain, ctf, csnet_ns, supdup, http_443, discard, gopher, daytime, sunrpc, efs, link, systat, exec, name, hostnames, mtp, echo, login, klogin, netbios_dgm, ldap, netstat, netbios_ns, netbios_ssn, ssh, kshell, nntp, sql_net, IRC, ntp_u, rje, remote_job, pop_2, X11, shell, printer, pm_dump, tim_i, urh_i, red_i, tftp_u, http_8001, aol, harvest, http_2784
	flag	SF, S0, REJ, RSTR, RSTO, S1, SH, S3, S2, RSTOS0, OTH
	src_bytes	0 - 1379963888
	dst_bytes	0 - 1309937401
	land	0, 1
	wrong_fragment	0 - 3
	urgent	0 - 3
Content features	hot	0 - 101
	num_failed_logins	0 - 5
	logged_in	0, 1

Continuation of Table 3.2

Type	Features	Range/Values
	num_compromised	0 - 7479
	root_shell	0 - 1
	su_attempted	0 - 2
	num_root	0 - 7468
	num_file_creations	0 - 100
	num_shells	0 - 5
	num_access_files	0 - 9
	num_outbound_cmds	0
	is_host_login	0, 1
	is_guest_login	0, 1
Time-based traffic features	count	0 - 511
	srv_count	0 - 511
	serror_rate	0 - 1
	srv_serror_rate	0 - 1
	rerror_rate	0 - 1
	srv_rerror_rate	0 - 1
	same_srv_rate	0 - 1
	diff_srv_rate	0 - 1
	srv_diff_host_rate	0 - 1
Connection-based traffic features	dst_host_count	0 - 255
	dst_host_srv_count	0 - 255
	dst_host_same_srv_rate	0 - 1
	dst_host_diff_srv_rate	0 - 1
	dst_host_same_src_port_rate	0 - 1
	dst_host_srv_diff_host_rate	0 - 1
	dst_host_serror_rate	0 - 1
	dst_host_srv_serror_rate	0 - 1
	dst_host_rerror_rate	0 - 1
	dst_host_srv_rerror_rate	0 - 1

### 3.3 Nominal Features Transformation using Probability Density Function (PDF)

Feature ranking and many classification algorithms are mathematical-based. Therefore, it is important to transform the nominal features of a dataset into their numerical representation. NSL-KDD dataset has three nominal features (as stated in Table 3.2): *protocol\_type*, *service*, and *flag*.

To avoid biasing the data, we did not encode the data with a static value map (e.g. http takes 1, smtp takes 2, and so on). Rather, we applied probability density function as in Eq. (3.1) [29] such that the most frequent nominal value in a column takes the highest numerical value while still being bounded between 0 and 1. This range goes along with the numerical features normalization (as explained in the next section).

$$PDF(x) = \frac{occur(x)}{n} \quad (3.1)$$

where  $occur(x)$  is the number of occurrences of value  $x$  within a column, and  $n$  is the total number of records.

### 3.4 Numerical Features Normalization using Min-Max

Another important step before working with feature ranking and classification algorithms is to normalize numeric features. Normalizing a feature means to scale its values to fall into a smaller range [36].

For example, there are features in NSL-KDD dataset that have wide range of values, such as: *duration*, *src\_bytes* and *num\_root*. While there are other features that have smaller range of values, such as: *num\_failed\_logins*, *is\_host\_login*, and *srv\_count*. Keeping the features without normalization may cause biasing towards selecting wide range features which may also affect classification performance.

To prevent this dominance, we chose to scale all the numeric features to fall in the range of  $[0, 1]$  using min-max normalization. Min-max scaling is shown in Eq. (3.2), where  $x$  is the value to be scaled in feature  $X$ ,  $MinMax(x)$  is the scaled value of  $x$ ,  $Min(X)$  and  $Max(X)$  are the minimum and maximum values respectively in

feature  $X$ ,  $min$  and  $max$  are the boundaries of the new range.

$$MinMax(x) = min + (max - min) \left( \frac{x - Min(X)}{Max(X) - Min(X)} \right) \quad (3.2)$$

### 3.5 Feature Ranking using ANOVA F-value

Feature ranking is a filter-based method that measures how significant a feature impacts the target class [75]. As indicated in Section 2.1, filter-based methods are usually statistical-based offering fast computation. However, they do not offer the ability to select the optimal subset of features. Therefore, we will use feature ranking to select a percentage<sup>1</sup> of the top ranked features to reduce the data dimension before passing it to a wrapper-based method that uses GWO as a search strategy. On the one hand, wrapper-based methods search for the optimal subset of features. On the other hand, they suffer from large search spaces which might take long time without giving the optimal subset. Therefore, this hybrid approach combines the best of both filter-based and wrapper-based methods.

One of the common statistical tests for feature ranking is Analysis of Variance (ANOVA). ANOVA uses F-value (or F-ratio) to measure whether the means of three or more groups are equal or not [76]. This is represented by its null hypothesis:  $H0 : \mu_1 = \mu_2 = \mu_3$  and its alternative hypothesis:  $H1 : \mu_1 \neq \mu_2 \neq \mu_3$ . If the null hypothesis is true, F-value is expected to be close to 1.

When applying ANOVA in univariate feature ranking, a method called one-way ANOVA is used. In this method, the F-value of each feature is independently computed, and the feature with the highest F-value is the top ranked feature. In other words, the top ranked feature has the most impact on the target class (i.e. groups). To better understand the F-value calculation, we will go through an example on NSL-KDD dataset.

In NSL-KDD dataset, each record is classified to one of five groups: Normal (no attack), DoS attack, Probing attack, R2L attack, and U2R attack. Assuming we are calculating the F-value of *service* and *flag*, and we have 31 records as in Table 3.3. Then, we have the following variables:

- $k$  = number of groups which is 5

---

<sup>1</sup>discussed alongside in section 5.2



- $N$  = number of records which is 31
- $N_{normal}, N_{dos}, N_{probing}, N_{r2l}, N_{u2r}$  = the number of records in each group which is: 9, 7, 6, 6 and 3 respectively.

Table 3.3: NSL-KDD ANOVA Example

Service	Flag	Category
21	2	Normal
32	10	Normal
39	60	Normal
45	60	Normal
57	60	Normal
52	60	Normal
48	60	Normal
33	60	Normal
32	60	Normal
11	25	DoS
14	60	DoS
13	10	DoS
19	10	DoS
22	25	DoS
25	10	DoS
30	25	DoS
3	60	Probing
7	60	Probing
4	60	Probing
9	10	Probing
5	2	Probing
6	10	Probing
5	60	R2L
3	2	R2L
2	60	R2L
3	60	R2L

Continuation of Table 3.3

Service	Flag	Category
4	10	R2L
8	60	R2L
3	2	U2R
3	60	U2R
2	60	U2R

The F-value of a variable  $X$  is the ratio of the variance between groups to the variance within groups [77]. This is represented in Eq. (3.3) where  $MS_{bg}$  is the mean squares between groups, and  $MS_{wg}$  is the mean squares within groups.

$$F = \frac{MS_{bg}}{MS_{wg}} \quad (3.3)$$

A mean squares is the weighted sum of square deviates divided to a degree of freedom. The degree of freedom is decided based on the context of the calculation. So, the degree of freedom of  $MS_{bg}$  (annotated  $df_{bg}$ ) is  $k - 1$ , and the degree of freedom of  $MS_{wg}$  (annotated  $df_{wg}$ ) is  $N - k$  as explained in Eq. (3.4).  $df_{bg}$  is called numerator degree of freedom, and  $df_{wg}$  is called denominator degree of freedom, and they are both used to draw the F-distribution  $F(df1, df2)$ . Figure 3.2 shows the F-distribution for  $df1$  of 4 and  $df2$  of 26 as in our example.

$$\begin{aligned}
 df_{wg} &= df_{normal} + df_{dos} + df_{probing} + df_{r2l} + df_{u2r} \\
 &= (N_{normal} - 1) + (N_{dos} - 1) + (N_{probing} - 1) + (N_{r2l} - 1) + (N_{u2r} - 1) \quad (3.4) \\
 &= N - k
 \end{aligned}$$

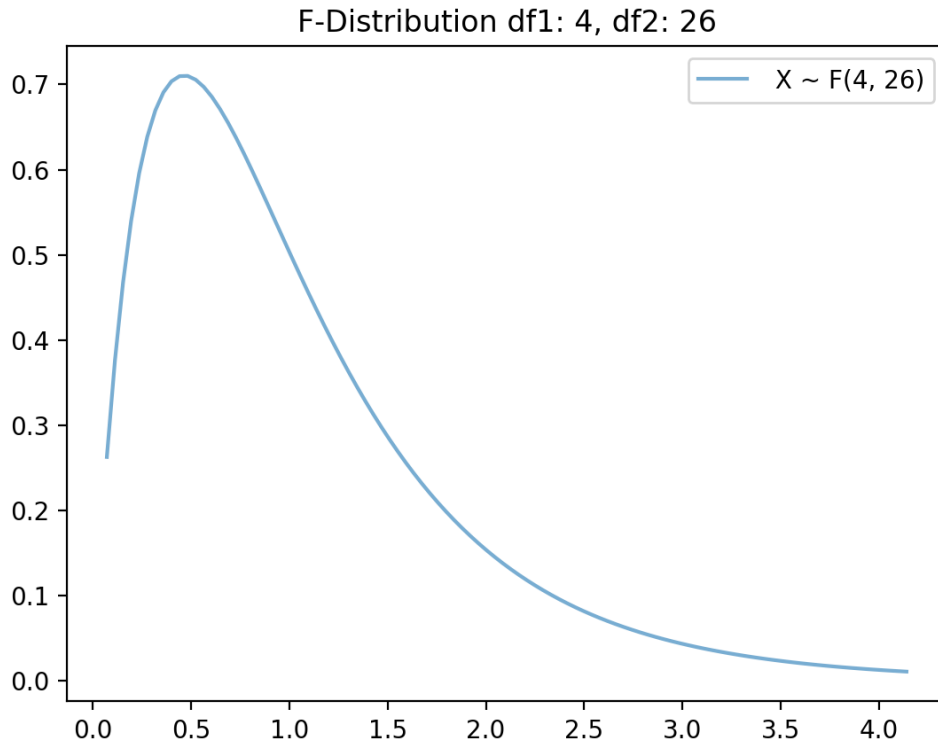
The weighted sum of square deviates between groups  $SS_{bg}$  and within groups  $SS_{wg}$  are given by Eq. (3.5) and Eq. (3.6):

$$SS_{bg} = \sum_{j=1}^k n_j (\bar{X}_j - \bar{X})^2 \quad (3.5)$$

$$SS_{wg} = \sum_{j=1}^k \sum (X - \bar{X}_j)^2 \quad (3.6)$$

where:

- $n_j$  = number of records in  $j^{\text{th}}$  group

Figure 3.2: F-Distribution  $F(4, 26)$ 

- $\bar{X}_j$  = sample mean of  $j^{\text{th}}$  group
- $\bar{X}$  = overall sample mean

Based on the above equations, we will use the following table to calculate F-value:

Table 3.4: F-value calculation template

Source of Variation	Sums of Squares (SS)	Degrees of Freedom (df)	Mean Squares (MS)	F
Between Groups	$SS_{bg}$	$k - 1$	$MS_{bg}$	$F = \frac{MS_{bg}}{MS_{wg}}$
Within Groups	$SS_{wg}$	$N - k$	$MS_{wg}$	

Correspondingly, Tables 3.5 and 3.6 show the F-value for *service* and *flag* features, respectively. Obviously, the F-value of *service* is much higher than the F-value

of *flag* in this example. This means that *service* has more impact than *flag* on the target class.

Table 3.5: F-value calculation for service feature

Source of Variation	Sums of Squares (SS)	Degrees of Freedom (df)	Mean Squares (MS)	F
Between Groups	$SS_{bg} = 7087.2916$	$k - 1 = 4$	$MS_{bg} = 1771.8229$	$F = 32.94$
Within Groups	$SS_{wg} = 1398.5794$	$N - k = 26$	$MS_{wg} = 53.7915$	

Table 3.6: F-value calculation for flag feature

Source of Variation	Sums of Squares (SS)	Degrees of Freedom (df)	Mean Squares (MS)	F
Between Groups	$SS_{bg} = 2586.4793$	$k - 1 = 4$	$MS_{bg} = 646.6198$	$F = 1$
Within Groups	$SS_{wg} = 16819.7143$	$N - k = 26$	$MS_{wg} = 646.9121$	

### 3.6 Feature Selection using BGWO

After applying feature ranking and selecting a percentage of the top ranked features, we will search for the optimal subset of features using a wrapper-based method that uses Grey Wolf Optimizer (GWO) [27] as a search strategy and Random Forest classifier<sup>2</sup> for the fitness function.

GWO is inspired from the social intelligence of grey wolf packs in leadership and hunting [28]. Grey wolves prefer to live in a pack with an average size between 5 and 12. Within a grey wolf pack, there is a social dominant hierarchy that consists of the following wolves [28] [23]:

<sup>2</sup>KNN classifier is proposed in the original algorithm, but we found that Random Forest is much faster and more accurate

1. Alpha wolves: these wolves are leading the pack in hunting, migration, feeding and decision making. Their decisions are dictated to the pack.
2. Beta wolves: these wolves are subordinate wolves that help the alpha in decision making or other activities.
3. Delta wolves: these wolves submit to alphas and betas.
4. Omega wolves: these wolves play the role of scape goat. They are the last wolves that are allowed to eat.

Mathematically, GWO tries to model the steps followed by the grey wolf pack in hunting a prey. These steps are [28]: chasing, encircling, harassing and attacking. Therefore, the fittest solution is called alpha  $\alpha$ , the second solution is beta  $\beta$  and the third solution is delta  $\delta$ . Other solutions are assumed to be omega  $\omega$ .

Encircling behavior is modeled by Eq. (3.7):

$$\vec{X}(t+1) = \vec{X}(t) - \vec{A} \cdot \vec{D} \quad (3.7)$$

where  $\vec{X}(t+1)$  is the next positions of a wolf,  $\vec{X}(t)$  is current position,  $\vec{A}$  is a coefficient matrix and  $\vec{D}$  is a vector that depends on the location of the prey  $\vec{X}_p(t)$  as shown in this equation:

$$\vec{D} = |\vec{C} \cdot \vec{X}_p(t) - \vec{X}(t)| \quad (3.8)$$

where  $\vec{C} = 2 \cdot \vec{r}_2$  and  $\vec{r}_2$  is a random vector in the range [0,1].

Eq. (3.7) can be simplified by making assumptions on the hunting behavior. The hunt is usually guided by alpha wolves. Beta and delta wolves have lower participation. These wolves are assumed to have better knowledge about the potential location of the prey. Therefore, their positions are considered the best three search agents, and other search agents (representing other wolves' positions) are obliged to update their positions based on these three best agents [23]. This can be simulated as in Eq. (3.9).

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3} \quad (3.9)$$

where  $\vec{X}_1$ ,  $\vec{X}_2$ ,  $\vec{X}_3$  are defined as in Eqs. (3.10) - (3.12), respectively.

$$\vec{X}_1 = |\vec{X}_\alpha - \vec{A}_1 \cdot \vec{D}_\alpha| \quad (3.10)$$

$$\vec{X}_2 = |\vec{X}_\beta - \vec{A}_2 \cdot \vec{D}_\beta| \quad (3.11)$$

$$\vec{X}_3 = |\vec{X}_\delta - \vec{A}_3 \cdot \vec{D}_\delta| \quad (3.12)$$

where  $\vec{X}_\alpha$ ,  $\vec{X}_\beta$ ,  $\vec{X}_\delta$  are the positions of  $\alpha$ ,  $\beta$ ,  $\delta$  at a given iteration  $t$  respectively.  $\vec{A}_1$ ,  $\vec{A}_2$ ,  $\vec{A}_3$  are defined as in Eq. (3.13), and  $\vec{D}_\alpha$ ,  $\vec{D}_\beta$ ,  $\vec{D}_\delta$  are vectors that depend on current positions and  $\vec{X}_\alpha$ ,  $\vec{X}_\beta$ ,  $\vec{X}_\delta$  and a random vector  $r_2$  in the range  $[0, 1]$ .

$$\vec{A} = 2\vec{a} \cdot \vec{r}_1 - \vec{a}[28] \quad (3.13)$$

In Eq. (3.13),  $\vec{r}_1$  is a random vector in the range  $[0,1]$ , and vector  $\vec{a}$  is linearly decreased from 2 to 0 over the course of iterations as shown in Eq. (3.14):

$$a = 2 - t \frac{2}{MaxIter} \quad (3.14)$$

where  $t$  is the iteration number and  $MaxIter$  is the total number of iterations allowed for the optimization.

From the mathematical model of GWO, it can be seen that GWO tries to make a balance between exploration and exploitation, which are two conflicting metrics [28]. Exploration is mainly controlled using  $\vec{C}$  vector, which is always in the range of  $[0, 2]$ . Since this vector is independent from the iteration number, it emphasizes exploration when there is a local optima. In addition to  $\vec{C}$ ,  $\vec{A}$  is also affecting the exploration. However,  $\vec{A}$  is affecting exploitation as well. This is mainly explained by the dependence of  $\vec{A}$  on the linearly decreasing vector  $\vec{a}$ . Since  $\vec{a}$  decreases from 2 to 0 based on the iteration,  $\vec{A}$  falls in the range of  $[-2,2]$ , and when it is larger than 1 or smaller than -1, exploration is emphasized, otherwise, exploitation is emphasized. The mathematical model of GWO also means that wolves are continuously changing their positions to whatever point in the space. This is not suitable for feature selection problems, where solutions should be in the binary form. Based on this need, Emary et al. [23] developed a binary version of GWO (BGWO).

In BGWO, all solutions are in binary form at any given time. This is achieved by applying a sigmoidal function on the product of  $\vec{A}$  and  $\vec{D}$  vectors in Eqs. (3.10)-(3.12). The sigmoidal function is applied as in Eq. (3.15) for  $\alpha$  solution and similarly for  $\beta$  and  $\delta$  solutions using  $\vec{A}_2 \vec{D}_\beta$  and  $\vec{A}_3 \vec{D}_\delta$  respectively.

$$cstep_\alpha = \frac{1}{1 + e^{-10(\vec{A}_1 \vec{D}_\alpha - 0.5)}} \quad (3.15)$$

After applying the sigmoidal function, a random threshold is applied as in Eq. (3.16).

$$bstep_{\alpha} = \begin{cases} 1 & \text{if } cstep_{\alpha} \geq rand \\ 0 & \text{otherwise.} \end{cases} \quad (3.16)$$

where  $rand$  is a random number drawn from uniform distribution of  $\in [0, 1]$ . The random threshold is applied similarly to  $\beta$  and  $\delta$  sigmoidal functions.

Then,  $\vec{X}_1$ ,  $\vec{X}_2$ ,  $\vec{X}_3$  are computed as in Eqs. (3.17) - (3.19).

$$\vec{X}_1 = \begin{cases} 1 & \text{if } (\vec{X}_{\alpha} + bstep_{\alpha}) \geq 1 \\ 0 & \text{otherwise.} \end{cases} \quad (3.17)$$

$$\vec{X}_2 = \begin{cases} 1 & \text{if } (\vec{X}_{\beta} + bstep_{\beta}) \geq 1 \\ 0 & \text{otherwise.} \end{cases} \quad (3.18)$$

$$\vec{X}_3 = \begin{cases} 1 & \text{if } (\vec{X}_{\delta} + bstep_{\delta}) \geq 1 \\ 0 & \text{otherwise.} \end{cases} \quad (3.19)$$

Finally, the updated position of all wolves  $\vec{X}(t+1)$  is computed using a suitable crossover between  $\vec{X}_1$ ,  $\vec{X}_2$  and  $\vec{X}_3$  as show in Eq. (3.20).

$$\vec{X}(t+1) = \begin{cases} \vec{X}_1 & \text{if } rand < \frac{1}{3} \\ \vec{X}_2 & \frac{1}{3} \leq rand < \frac{2}{3} \\ \vec{X}_3 & \text{otherwise} \end{cases} \quad (3.20)$$

where  $rand$  is a random number drawn from uniform distribution in the range  $[0, 1]$ .

In BGWO, the best feature combination is the one with maximum classification performance (or minimum error rate) and minimum number of selected features. Therefore, the fitness function used in BGWO is shown in (3.21).

$$Fitness = \alpha E_R(D) + \beta \frac{|R|}{|C|} \quad (3.21)$$

where  $E_R(D)$  is the error rate for the classifier of condition attribute set,  $R$  is the length of selected feature subset, and  $C$  is the total number of features,  $\alpha$  and  $\beta$  are two parameters corresponding to the importance of classification quality and subset length,  $\alpha \in [0, 1]$  and  $\beta = 1 - \alpha$ ;  $\beta = 0.01$  by author [27] experiments.

### 3.7 Multi-class Classification using One-vs-rest Strategy

Intrusion detection is a multi-class classification problem [3]. There are two common strategies for such problems: one-vs-one and one-vs-rest (OvR). In one-vs-one, a binary classifier is trained for each pair of classes, resulting in a total of  $\frac{N(N-1)}{2}$  binary classifiers where  $N$  is the number of classes. The class of a new test instance is determined based on a majority vote. While this strategy involves smaller subsets passed to each classifier, it is still considered computationally slower than one-vs-rest strategy. Hence, we adopted one-vs-rest strategy.

In one-vs-rest, the training dataset is passed to  $N$  binary classifiers each representing a class. As shown in Figure 3.3, each classifier takes the data with two labels: positive label (usually 1) representing the assigned class, and negative (or zero) label representing any other class.

When it comes to prediction, each classifier estimates a probability of classifying the new test instance to its assigned class. The highest probability determines the class of the test instance. This is illustrated in Figure 3.4 [78].



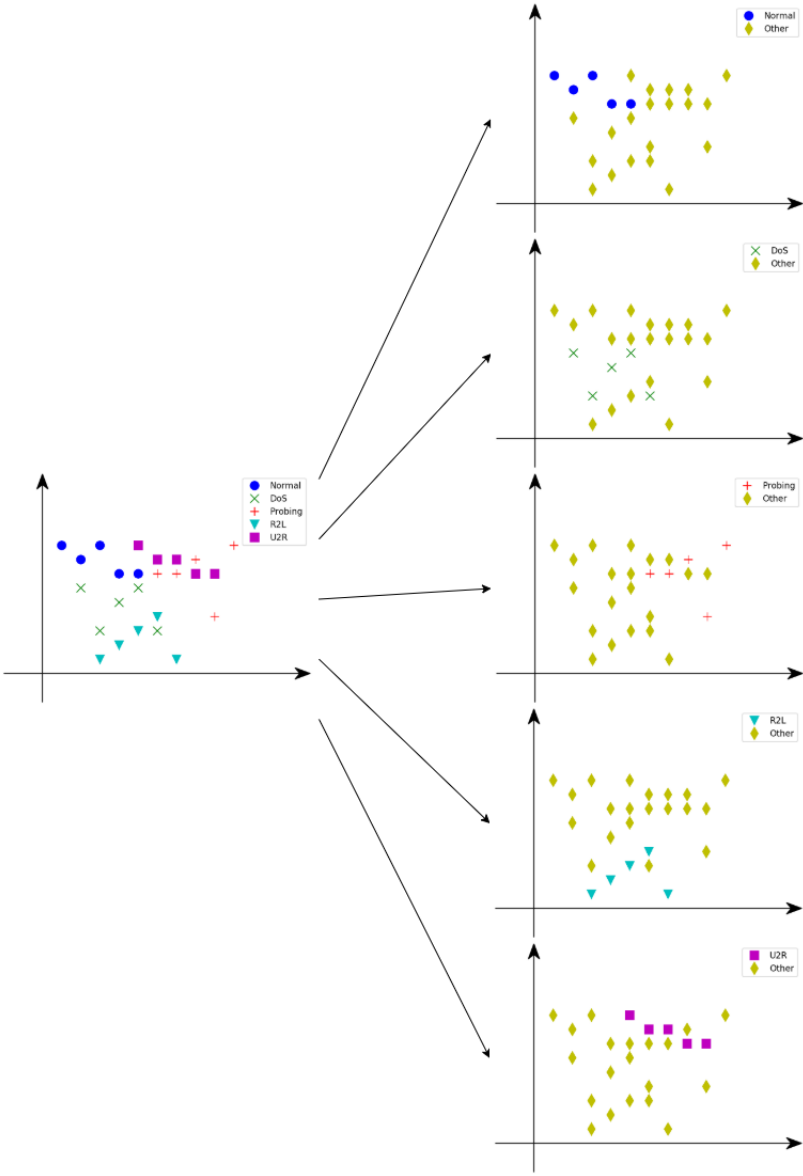


Figure 3.3: One-vs-rest (OvR) Dataset Separation

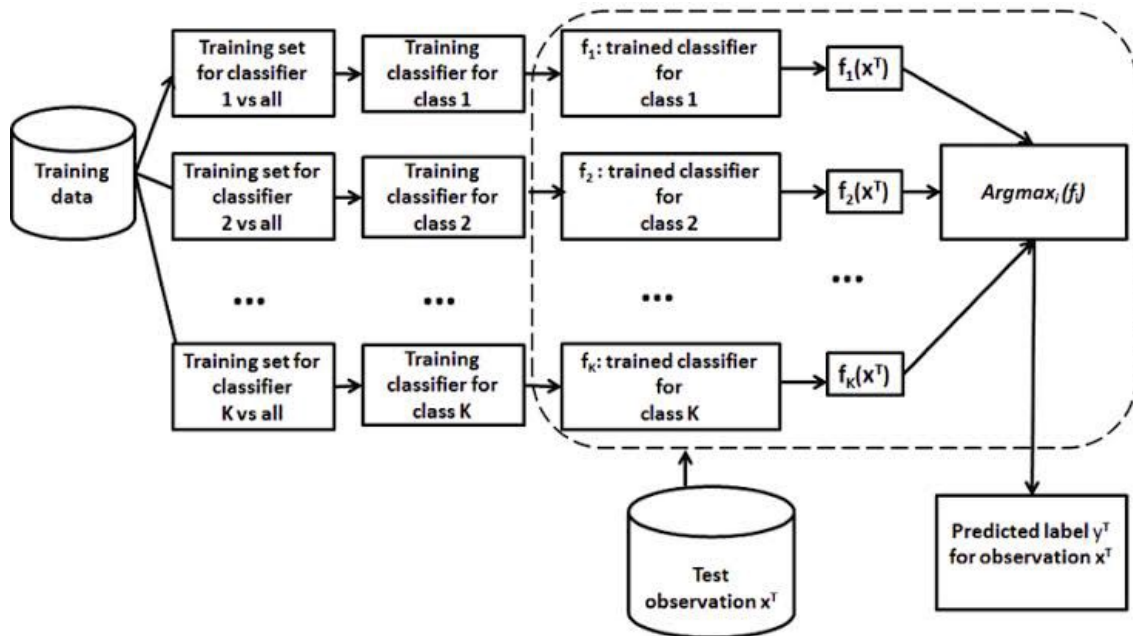


Figure 3.4: One-vs-rest (OvR) Approach

### 3.8 Summary

In this chapter, we first presented an overview of the proposed approach. There are three main stages:

1. Splitting the dataset in a stratified fashion into 70% training dataset and 30% testing dataset.
2. Preprocessing: starts by transforming the nominal features to numerical features using probability density function (PDF), and then numerical features are normalized using min-max, and at the last stage feature selection using the proposed approach is applied.
3. Classification

After that, we analyzed NSL-KDD dataset to understand the motivation behind it, the main attacks it contains, and its features with their respective values/range. Then, we briefly explained nominal features transformation and numerical features normalization.

After that, we explained thoroughly ANOVA F-value with an example. Simply saying, ANOVA F-value is a well-known statistical test that tests whether the means of three or more groups are equal or not. If F-value is around 1, then the means are almost equal. If F-value is very large, then the means are different. When applying ANOVA F-value to feature selection, we are aiming to select the features that have the highest F-value. Because this means that these features can be used to predict the class label.

After that, we explained thoroughly the mathematical model behind GWO and BGWO. We also emphasized how mathematically GWO tries to balance between exploration and exploitation. GWO, as a result, can efficiently manage the trade-off between local optima and global optima. This is a main characteristic in any efficient metaheuristic algorithm.

Finally, we presented a strategy for multi-class classification called one-vs-rest (OvR). In this strategy, there will be a binary classifier for each label. In each classifier, the positive label is for the assigned label, and the negative label is for any other label. The training data will be passed to each of these classifiers. When

we want to predict a testing record, each classifier gives a probability on classifying that record into its positive label. The label of the classifier with highest probability will be assigned to the testing record.

# Chapter 4

## Experimental Setup and Evaluation Metrics

### 4.1 Methodology

The proposed approach implements a hybrid feature selection algorithm that uses ANOVA F-value to reduce the features dimension before applying a wrapper method. The wrapper method uses GWO as a search strategy. The main goals of the proposed approach are:

1. To reduce classification training and prediction time, while achieving similar or better classification performance.
2. To reduce the search space and computation complexity of GWO such that faster convergence is achieved and a smaller subset of features is selected.

Before verifying the efficiency of the proposed method, we need to tune the main parameters affecting its performance. These parameters are:

1. ANOVA F-value Threshold: the threshold in this case is the percentage of top-ranked features that we want to select.
2. GWO Population size: affects the convergence speed of GWO.
3. GWO Iterations: affects the execution time of GWO.

After that, the classification time and performance of the proposed approach are compared to the classification time and performance with GWO only (without

ANOVA F-value) and without feature selection. To evaluate the classification performance, we chose five classification models that are commonly used in IDS [37] [79]: SVM, KNN, Decision Tree, Naive Bayes, and ANN.

For the sake of this research, we ran the experiments on NSL-KDD dataset, but we plan to continue experimentation on other recent datasets, such as CIC-IDS-2017 and CSE-CIC-IDS-2018. To make sure that we avoid any overfitting or biasing issues, we ran 30 experiments for each of the three approaches. This applies also to the parameters tuning experiments. In each experiment run, the data is randomly shuffled and separated to 70% training set and 30% testing set. The separation is performed in a stratified fashion, so it preserves the original data distribution. Afterwards, we consider the average of each evaluation metric to do the comparison. The results and their analysis are reported in the next chapter.

## 4.2 Environment and Implementation Tools

We ran the experiments on a machine with the following specifications:

Table 4.1: Machine specs

OS	Windows 10 Pro 64-bit
RAM	16.0 GB
CPU	Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz with 4 Cores and 8 Logical Processors

Furthermore, the algorithm was implemented in Python 3.7.5 where the following tools and libraries were used:

- PyCharm Community Edition: a well-known IDE for Python applications.
- NumPy [80]: a numerical Python library that makes it easy to manipulate arrays and matrices.
- pandas [81]: a Python library used for data manipulation and analysis. So it makes it easy to read/write the data from/to CSV or Excel files.
- Matplotlib [82]: a plotting Python library that works well with pandas and numpy.

- BGWO Matlab implementation [23]: the original implementation of the algorithm from the authors. We did the same implementation in Python taking advantage of EvoloPy [83], which is a repository that implements many SI algorithms (including GWO) in Python.
- scikit-learn [84]: a well-known machine-learning library. We used many out of the box algorithms from this library including:
  - *f\_classif* to realize ANOVA F-value and *SelectPercentile* to apply a threshold on top ranked features.
  - *SVC* which stands for Support Vector Classifier is an SVM based algorithm that by default uses radial basis function (RBF) as a kernel.
  - *KNeighborsClassifier* with the default value of neighbors which is 5.
  - *DecisionTreeClassifier* with the default algorithm which is CART (Classification and Regression Tree).
  - *GaussianNB* stands for Gaussian Naive Bayes.
  - *MLPClassifier* stands for Multi-layer Perceptron classifier, which is an ANN algorithm. It comes with a default value of 100 for the hidden layers.
  - *OneVsRestClassifier* which is a realization for one-vs-rest multi-class classification strategy.

### 4.3 Evaluation Metrics

The effectiveness of any IDS is mainly measured by [3] [58] [73]: overall accuracy, detection rate, false alarm rate, training time, and prediction time. A well-performing IDS would achieve a low false alarm rate, and high accuracy and detection rate.

The common way to derive the definition of these metrics is through a confusion matrix. Just for simplicity, we will assume an IDS that performs binary classification. In this IDS, a record that is labeled as "attack" is a "Positive" record, and a record that is labeled as "normal" is a "Negative" record. Confusion matrix is a two

by two matrix that represents the four possible combinations of the actual records and the predicted records.

Table 4.2: Confusion matrix

		Predicted	
		Negative (normal)	Positive (attack)
Actual	Negative (normal)	$TN$	$FP$
	Positive (attack)	$FN$	$TP$

Table 4.2 shows a confusion matrix where:

- True Negative (TN): represents the number of normal records correctly predicted as normal.
- False Positive (FP): represents the number of attack records wrongly predicted as normal.
- False Negative (FN): represents the number of normal records wrongly predicted as attack.
- True Positive (TP): represents the number of attack records correctly predicted as attack.

Based on the confusion matrix, we can define the metrics mentioned above as follows:

- Overall Accuracy: is the percent of correctly classified records. It is calculated by:

$$\text{Overall Accuracy} = \frac{TN + TP}{TN + FP + FN + TP} \quad (4.1)$$

- Detection Rate (DR): also called Recall or sensitivity or true positive rate (TPR). It is the percent of correctly classified attacks to the total number of actual attacks. When it is near 1, it means that the classifier performed well in predicting almost all actual attacks. It is calculated by:

$$\text{Detection Rate (DR)} = \frac{TP}{TP + FN} \quad (4.2)$$

- False Alarm Rate (FAR): also called False Positive Rate (FPR). It is the percentage of wrongly classified normal records. When it is near zero, it means



that the classifier performed well in avoiding misprediction of almost all normal records. It is calculated by:

$$FAR = \frac{FP}{FP + TN} \quad (4.3)$$

Another two well-known metrics that can be calculated from confusion matrix are Prediction and F1. They are calculated as in Eq. (4.4) and Eq. (4.5).

$$Precision = \frac{TP}{TP + FP} \quad (4.4)$$

$$F1 = \frac{2TP}{2TP + FP + FN} \quad (4.5)$$

In this study, we are considering intrusion detection as a multi-class problem. In multi-class case, all metrics except overall accuracy need to be calculated per class. Therefore, we have to treat each class as if it is the positive class that we want to detect, and the other classes are negative. We demonstrate this treatment by having an example of a 5x5 confusion matrix on NSL-KDD classes. This confusion matrix is shown in Table 4.3.

Table 4.3: Multi-class confusion matrix

		Predicted				
		Normal	DoS	Probe	R2L	U2R
Actual	Normal	21761	286	677	287	106
	DoS	1183	14535	209	74	15
	Probe	510	72	3550	68	23
	R2L	631	77	197	235	24
	U2R	31	1	1	2	1

We will break this confusion matrix into 5 smaller matrices each of size 2x2. Figure 4.1 shows how to calculate TP, FP and FN when we consider U2R class. TN is calculated by subtracting (TP, FP, FN) from the sum of the matrix which is 44556 in this case. Same idea applies on each class.

Tables 4.4 - 4.8 show the resulting confusion matrices for each class in NSL-KDD. And now, we can easily calculate the metrics as follows:

- Overall Accuracy:  $\frac{21761+14535+3550+235+1}{44556} = \frac{40082}{44556} = 0.8996$

		Predicted				
		Normal	DoS	Probe	R2L	U2R
Actual	Normal	21761	286	677	287	106
	DoS	1183	14535	209	74	15
	Probe	510	72	3550	68	23
	R2L	631	77	197	235	24
	U2R	31	1	1	2	1

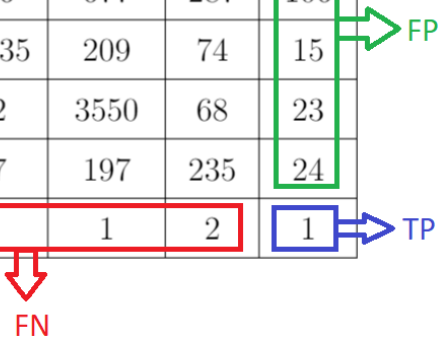


Figure 4.1: Example on how to calculate TP, FN and FP for U2R class

- Normal DR:  $\frac{21761}{21761+1356} = \frac{21761}{23117} = 0.9413$
- DoS DR:  $\frac{14535}{14535+1481} = \frac{14535}{16016} = 0.9075$
- Probe DR:  $\frac{3550}{3550+673} = \frac{3550}{4223} = 0.8407$
- R2L DR:  $\frac{235}{235+929} = \frac{235}{1164} = 0.2019$
- U2R DR:  $\frac{1}{1+35} = \frac{1}{36} = 0.0278$
- Normal FAR:  $\frac{2355}{2355+19084} = \frac{2355}{21439} = 0.1098$
- DoS FAR:  $\frac{436}{436+28104} = \frac{436}{28540} = 0.0153$
- Probe FAR:  $\frac{1084}{1084+39249} = \frac{1084}{40333} = 0.0269$
- R2L FAR:  $\frac{431}{431+42961} = \frac{431}{43392} = 0.0099$
- U2R FAR:  $\frac{168}{168+44352} = \frac{168}{44520} = 0.0037$

Table 4.4: Normal Confusion matrix

		Predicted	
		Other	Normal
Actual	Other	19084	2355
	Normal	1356	21761

Table 4.5: DoS Confusion matrix

		Predicted	
		Other	DoS
Actual	Other	28104	436
	DoS	1481	14535

Table 4.6: Probe Confusion matrix

		Predicted	
		Other	Probe
Actual	Other	39249	1084
	Probe	673	3550

Table 4.7: R2L Confusion matrix

		Predicted	
		Other	R2L
Actual	Other	42961	431
	R2L	929	235

Table 4.8: U2R Confusion matrix

		Predicted	
		Other	U2R
Actual	Other	44352	168
	U2R	35	1

# Chapter 5

## Result Analysis

In this chapter, we first present how we selected the population and iteration for GWO, and then we conclude the performance analysis of our hybrid feature selection approach.

### 5.1 Parameters Tuning for GWO

There are two main factors affecting the convergence and search time of GWO: population and maximum iterations. Population refers to the number of wolves in GWO and it is the number of solutions. Maximum iterations is used as a stopping criteria in GWO.

Since we are using BGWO, which is the binary version of GWO, it is stated in the original paper [23] that BGWO achieves fast convergence with reasonable population size and within few iterations. To verify these findings, we choose to perform our experiments using different combinations of population and iteration as shown in in Table 5.1. Furthermore, we included all the possibilities of ANOVA F-value thresholds (percentages of top ranked features). This is to make sure that we are not doing a biased measurement, and it will be useful to select the most reasonable ANOVA F-value threshold later on.

Table 5.1: Chosen values for ANOVA F-value threshold, GWO population, and GWO iterations

ANOVA F-value	20%, 25%, 30%, 35%, 40%, 45%, 50%, 55%, 60%, 65%, 70%
GWO Population	5, 10, 20
GWO Iteration	10, 20, 40, 70

In terms of convergence, we plotted for each population four graphs, each representing an iteration value. This is shown in Figures 5.1-5.3, where each line represents the average value of fitness for each iteration based on a specific threshold from ANOVA F-value. It can be seen from these figures that 10 iterations are more than enough for the fitness to converge regardless of the threshold value. The same can be observed from the selected features per iteration graphs (Figure 5.4-5.6). Moreover, Figures 5.7-5.9 show that having more than 10 iterations will highly affect the execution time of BGWO without actually having better convergence.

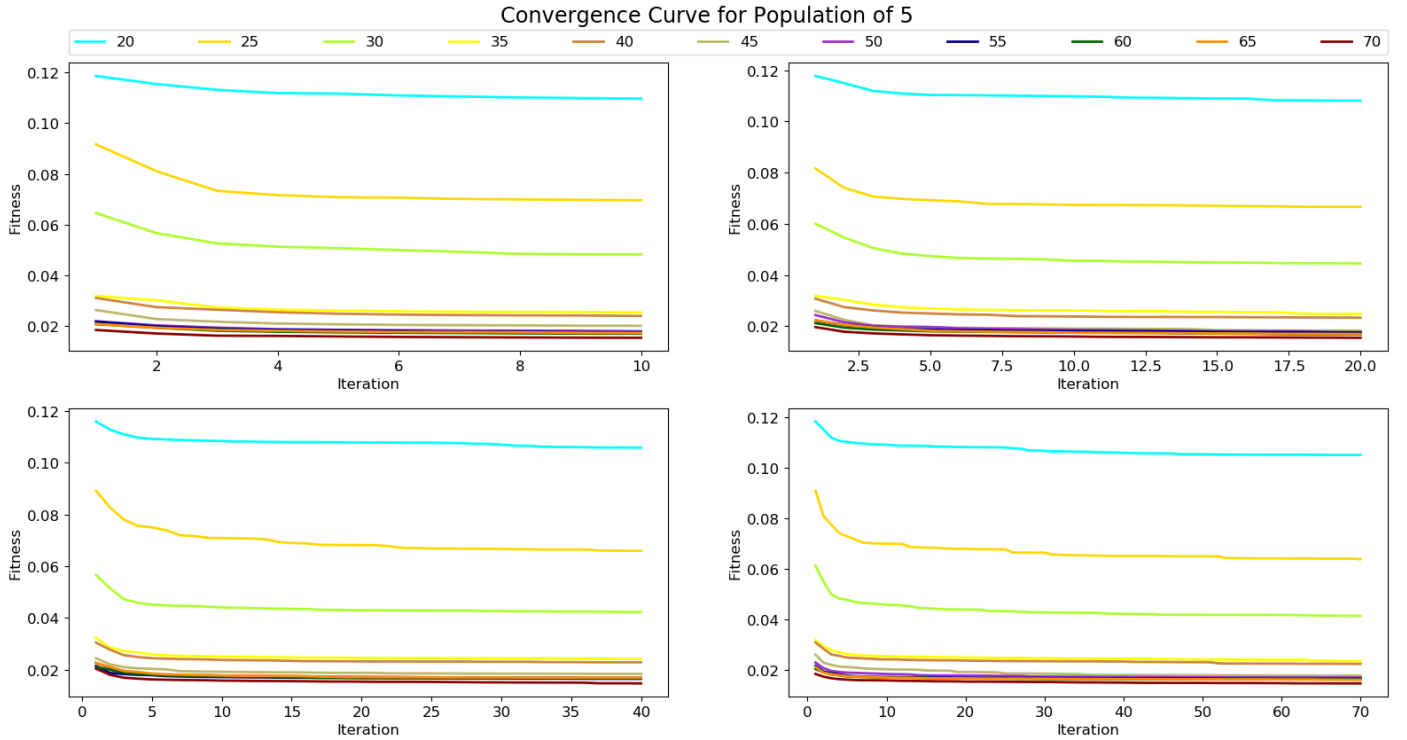


Figure 5.1: GWO Convergence curve when Population is 5

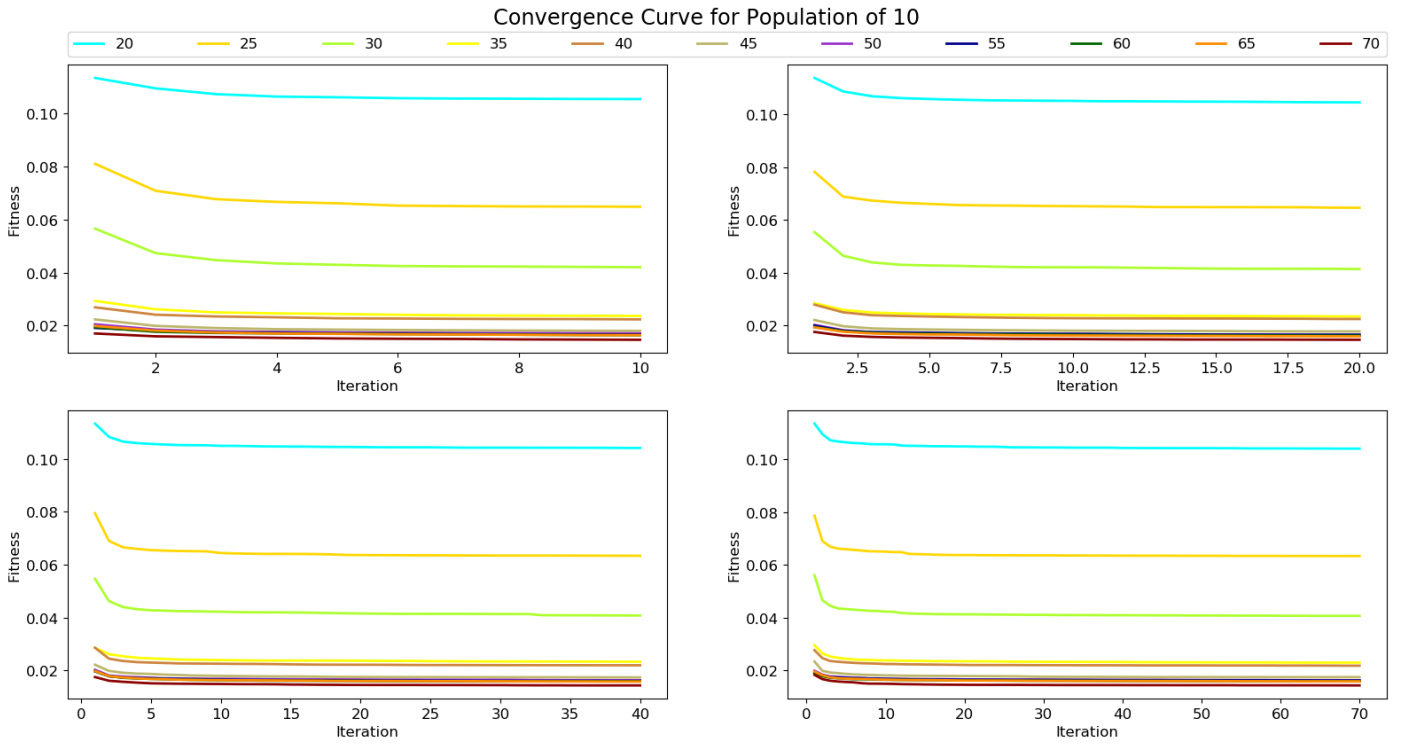


Figure 5.2: GWO Convergence curve when Population is 10

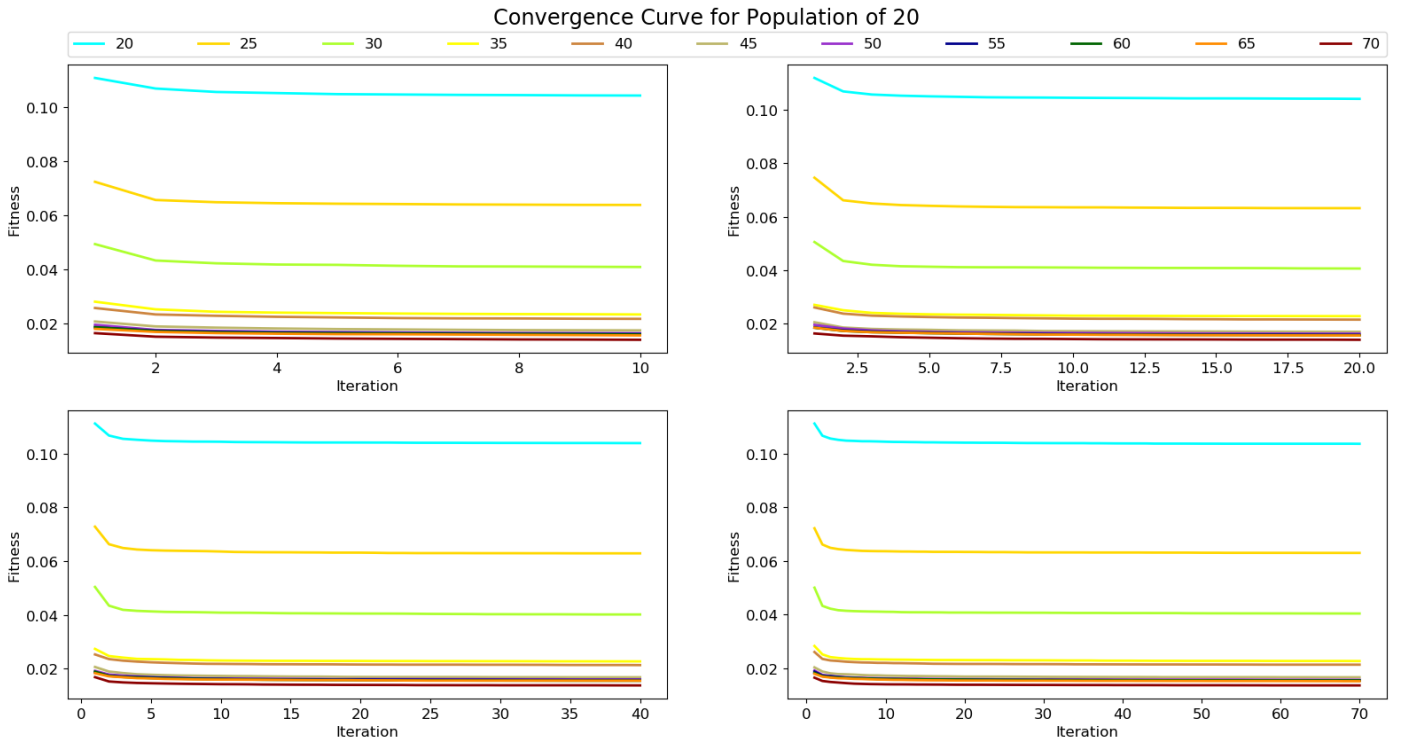


Figure 5.3: GWO Convergence curve when Population is 20

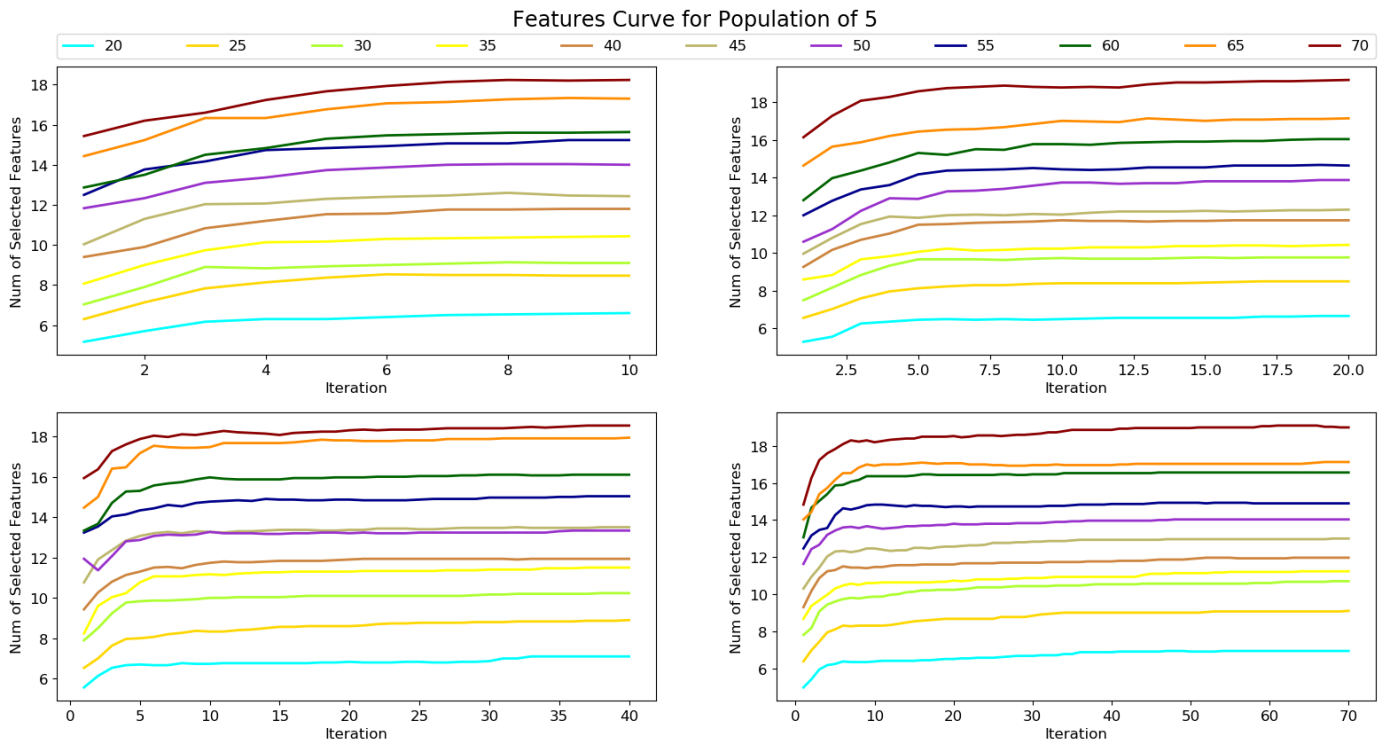


Figure 5.4: GWO Selected Features curve when Population is 5

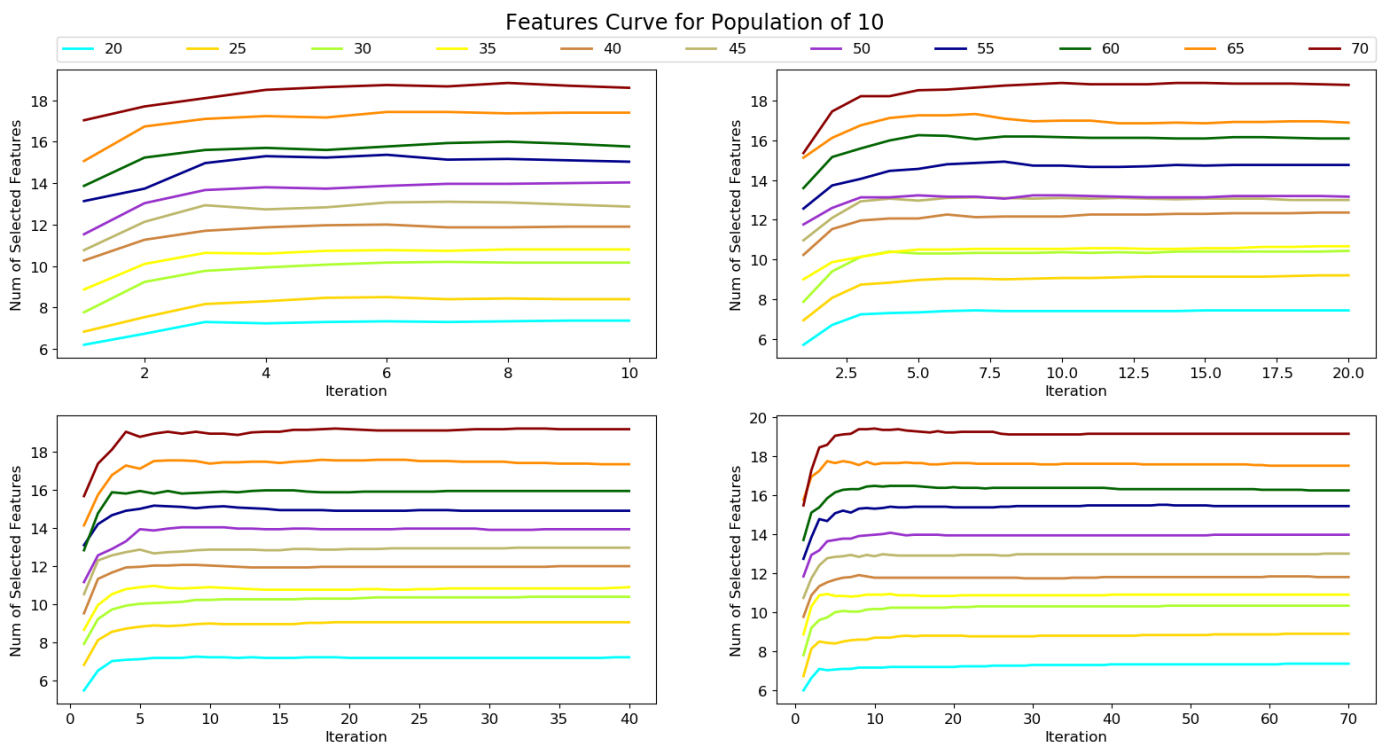


Figure 5.5: GWO Selected Features curve when Population is 10

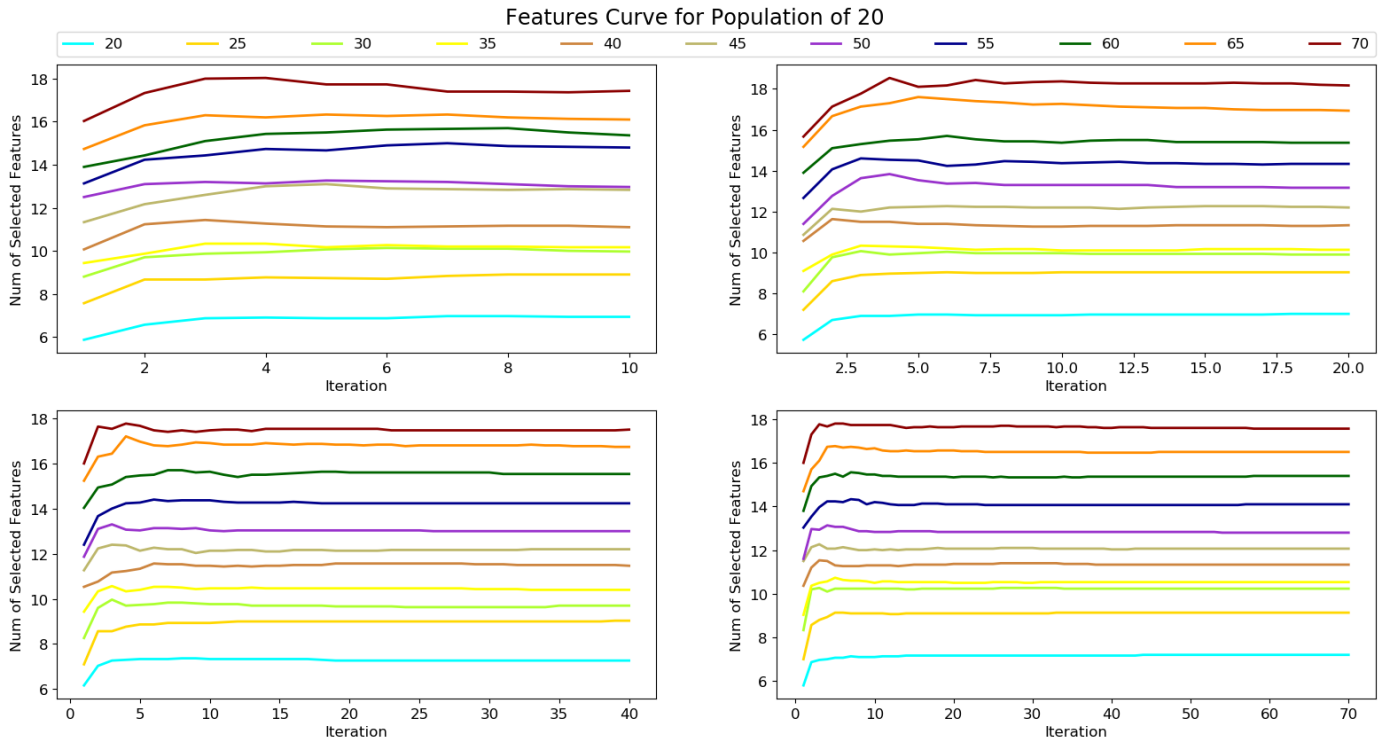


Figure 5.6: GWO Selected Features curve when Population is 20

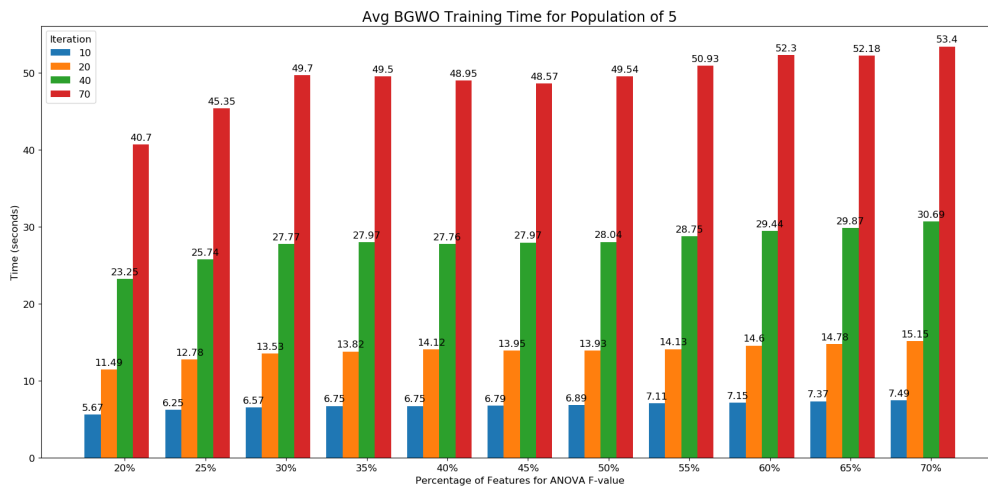


Figure 5.7: GWO Avg Training Time when Population is 5



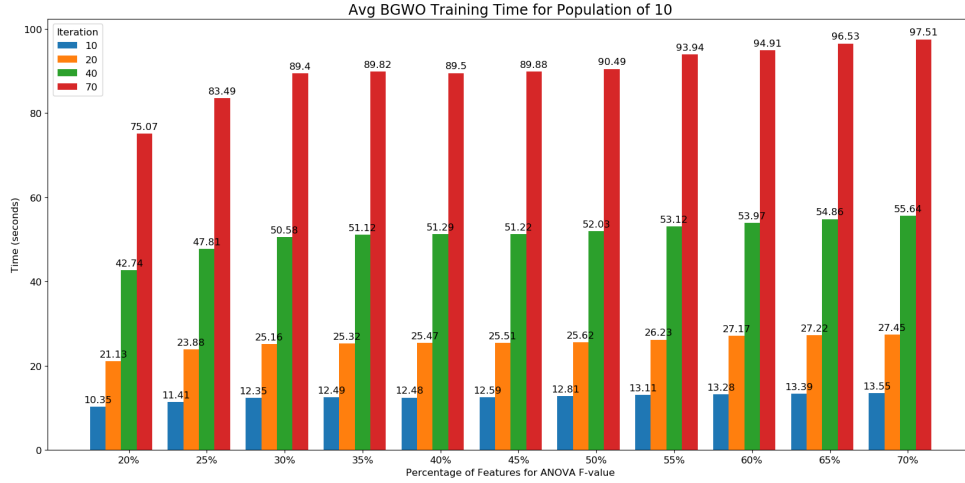


Figure 5.8: GWO Avg Training Time when Population is 10



Figure 5.9: GWO Avg Training Time when Population is 20

So, the suitable maximum iteration is 10. To determine the suitable population for this number of iterations, we need to compare the starting and ending values of the fitness. It is difficult to check these values from Figures 5.1-5.3. Therefore, we included them separately in Table 5.2. From this table, it can be noticed that a population size of 20 does always have the lowest starting and ending values of the fitness regardless of the threshold.

Table 5.2: The starting and ending values of the fitness with 10 iterations

ANOVA F-value Threshold (Percentage)	Population	Starting Fitness Value	Ending Fitness Value
20	5	0.11863	0.10972
	10	0.11345	0.10548
	20	<b>0.11065</b>	<b>0.10415</b>
25	5	0.09167	0.06967
	10	0.08108	0.06486
	20	<b>0.07230</b>	<b>0.06376</b>
30	5	0.06462	0.04832
	10	0.05659	0.04202
	20	<b>0.04929</b>	<b>0.04084</b>
35	5	0.03191	0.02541
	10	0.02931	0.02364
	20	<b>0.02803</b>	<b>0.02331</b>
40	5	0.03115	0.0241
	10	0.02689	0.02233
	20	<b>0.0257</b>	<b>0.02169</b>
45	5	0.0264	0.02021
	10	0.02232	0.018
	20	<b>0.02071</b>	<b>0.01743</b>
50	5	0.022	0.01806
	10	0.02049	0.01703
	20	<b>0.0195</b>	<b>0.01635</b>
55	5	0.02182	0.01761
	10	0.01912	0.01656
	20	<b>0.01857</b>	<b>0.01611</b>
60	5	0.02086	0.01707
	10	0.0194	0.01636
	20	<b>0.01861</b>	<b>0.01589</b>

Continuation of Table 5.2

ANOVA F-value Threshold (Percentage)	Population	Starting Fitness Value	Ending Fitness Value
65	5	0.02092	0.01736
	10	0.01984	0.01639
	20	<b>0.01799</b>	<b>0.01557</b>
70	5	0.01854	0.01547
	10	0.01701	0.01464
	20	<b>0.01641</b>	<b>0.01396</b>

## 5.2 Classification Performance and Selected Features

After choosing the population size as 20 and the maximum iterations as 10 for GWO, we will compare the performance of our hybrid feature selection approach with: the whole process without ANOVA F-value, and the whole process without feature selection. As in the previous section, we present the different combinations of the ANOVA F-value thresholds (see Table 5.1) within our approach.

As a matter of fact, we have noticed that the classifiers of Decision Tree, MLP, and KNN are already performing well without any feature selection in terms of overall accuracy, detection rate and false alarm rate for most of the classes. While SVC achieved lower overall accuracy, it achieved higher detection rate for some classes. On the other hand, Gaussian NB was the worst among these, especially when it comes to overall accuracy.

Consequently, our main goal will be to emphasize that using a lower number of features, we have been able to reduce the training time and testing time of most of the classifiers while achieving almost the same metrics (less in most cases). At the same time, we will highlight the reduction in GWO search time.

Throughout our experiments, we have concluded that the threshold of 50% for

ANOVA F-value is the most balanced threshold. At this threshold, we are able to say that there is a good reduction in number of features, training time, and testing time, while achieving almost the same metrics.

Also at this threshold, the features are first reduced from 41 to 20 in ANOVA F-value (see Figure 5.10), and then it is reduced to an average of 13 feature in GWO. Figure 5.11 shows the how frequent features were selected during the thirty experiments.

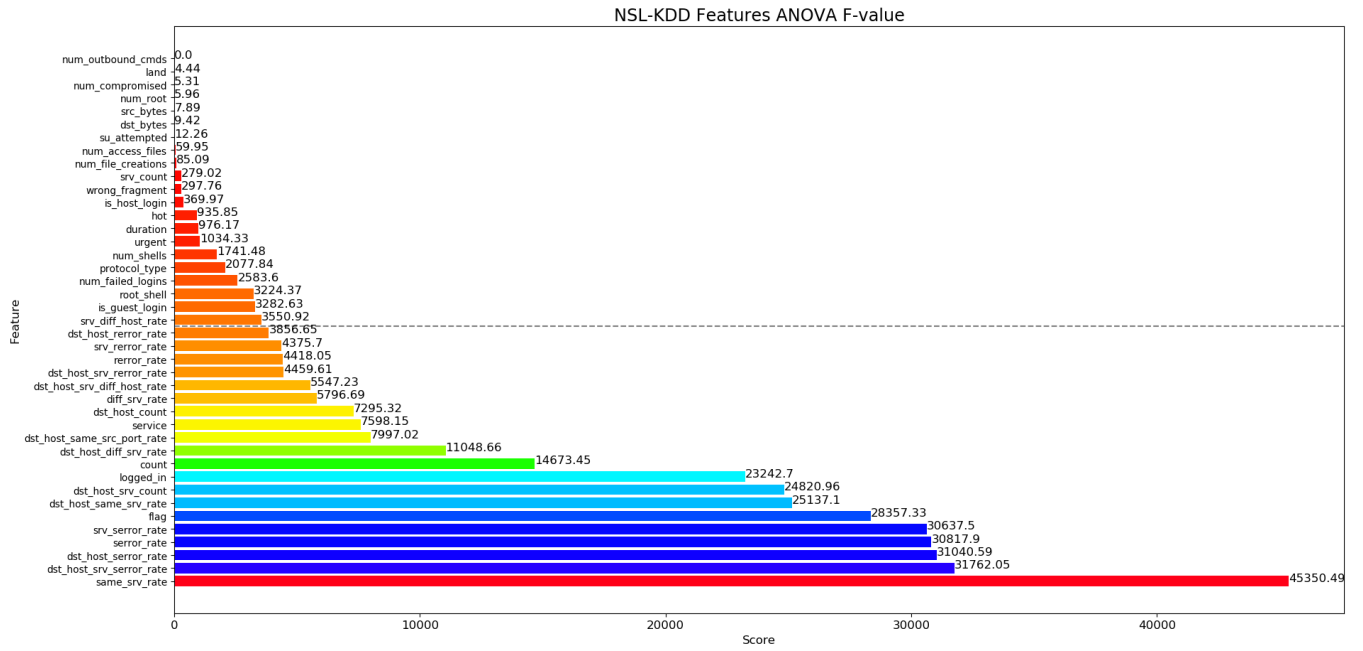


Figure 5.10: Average ANOVA F-value for each feature in NSL-KDD. All the 20 features below the horizontal grey line were selected

Compared to GWO only without any prior feature ranking, we can see that the approach method reduces the search time by five seconds, as shown in Table 5.3.

Table 5.3: GWO with 50% threshold vs. GWO only

Approach	Avg Selected Features	Search Time (seconds)
GWO with 50% threshold	13	22.778
GWO without threshold	25	27.029

Based on the threshold of 50%, we start with Figures 5.12a and 5.12b which show the training time and testing time of the different classifiers. We can see that some classifiers, such as Decision Tree and Gaussian NB are already fast even without

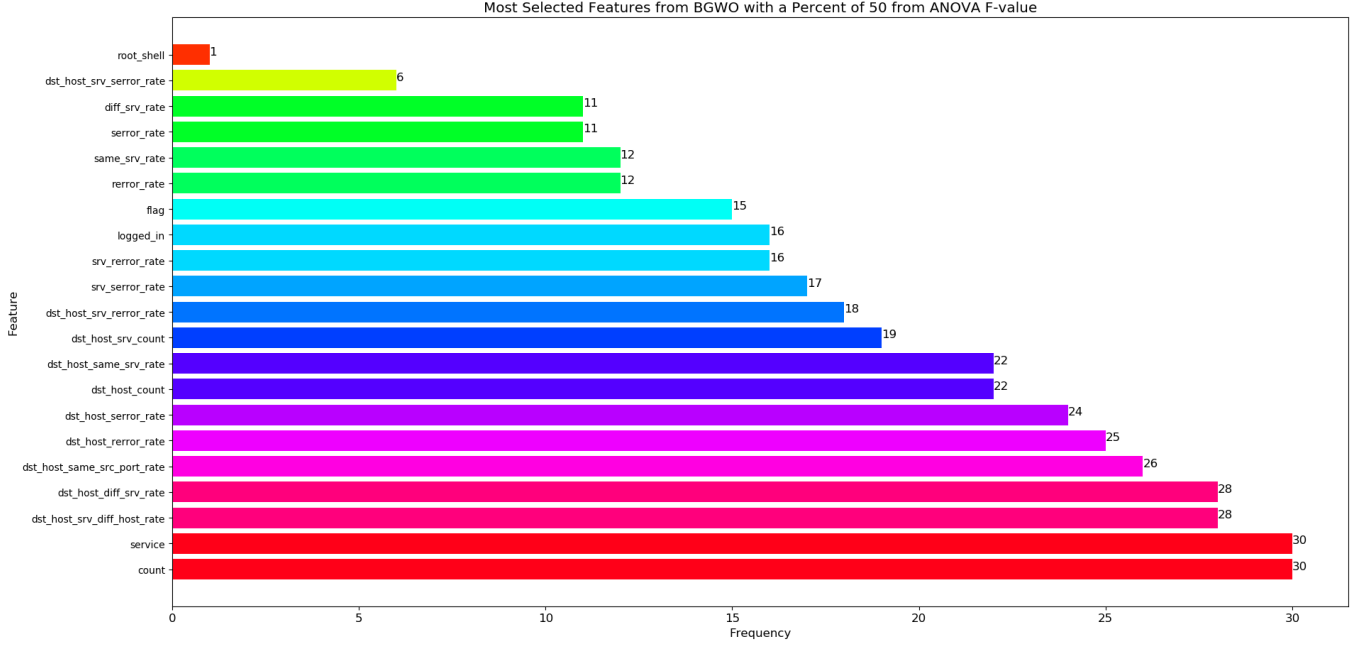


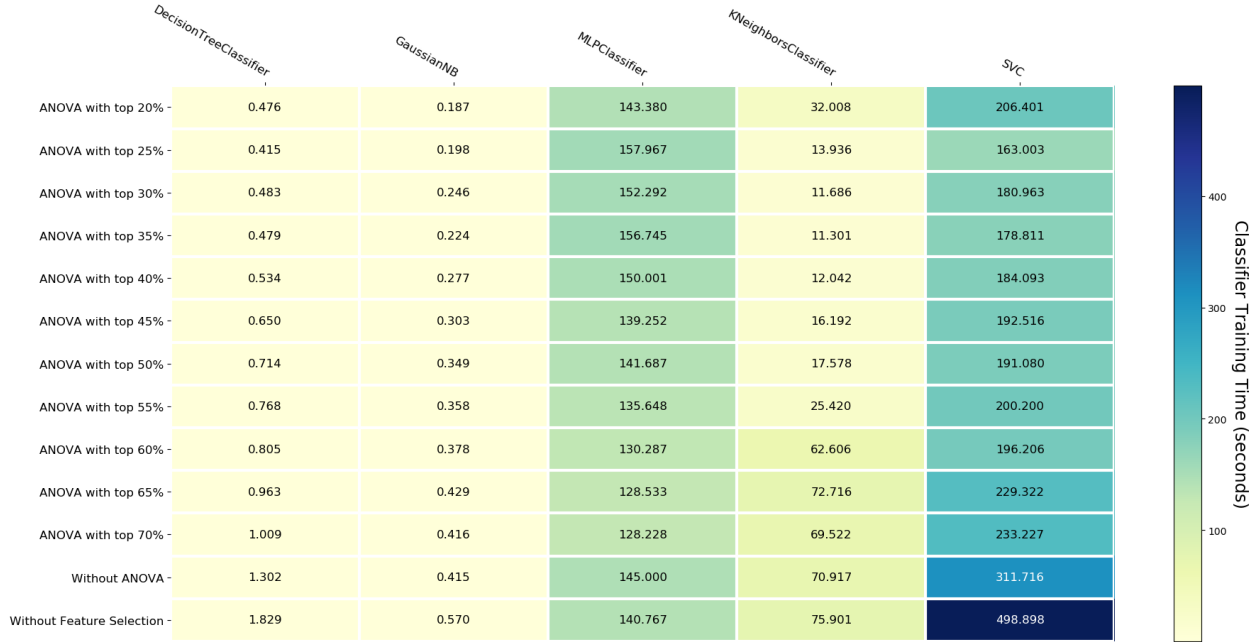
Figure 5.11: Most frequent features selected by GWO with a threshold of 50%

any feature selection. While for KNN and SVM, the training time and the testing time are significantly improved with the selected threshold. The training time is reduced from 75.901 seconds to 17.578 seconds in case of KNN (77% reduction), and from 498.898 seconds to 191.080 seconds in case of SVM (62% reduction). And the testing time is reduced from 165.024 seconds to 11.592 seconds in case of KNN (93% reduction), and from 72.116 seconds to 39.157 seconds in case of SVM (46% reduction).

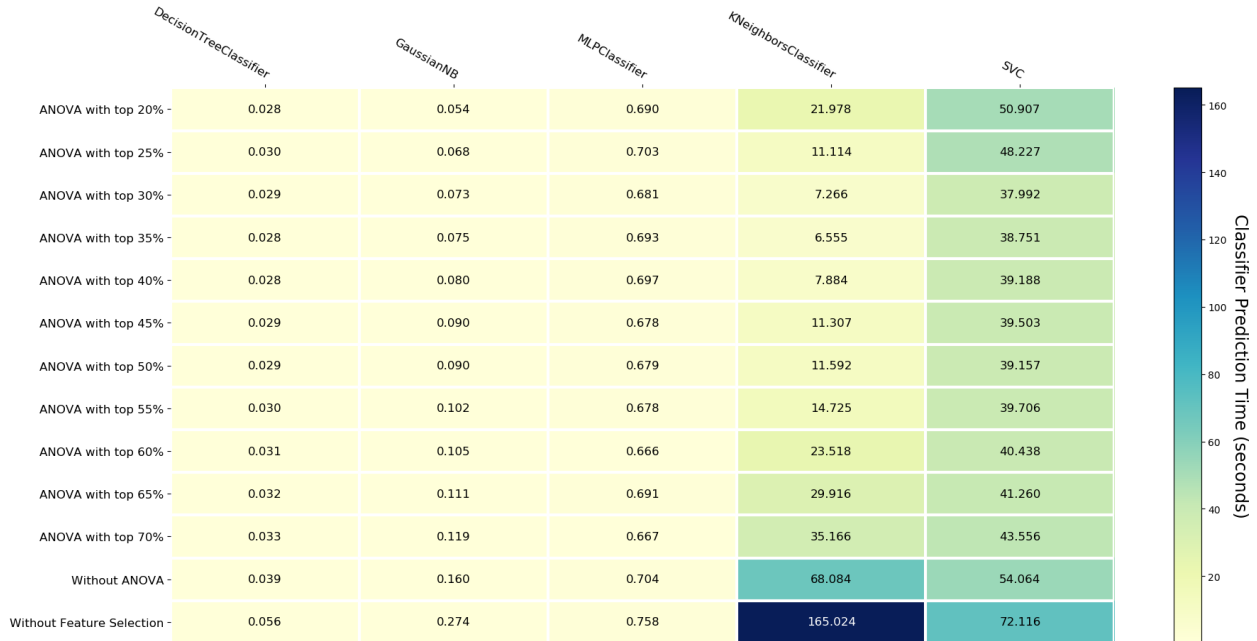
If we look at the overall accuracy in Figure 5.13, we can observe that the accuracy of Gaussian NB was improved using the selected threshold from 36.434% to 81.593%. For the other classifiers, the accuracy without feature selection is a bit higher, but still we are able to achieve almost the same accuracy.

When it comes to detection rate of each class (shown in Figures 5.14-5.18), we can see that there is a similar behavior in Normal and DoS classes DRs to the behavior we have seen in overall accuracy. This is because these two classes are the majority, so detecting them is an easier task. While for Probe class, we are getting similar detection rate except for Gaussian classifier which have a much higher value. And finally, regarding R2L and U2R classes, these two are the most difficult to detect in any IDS and using any method because of their low number of records in the dataset.

When it comes to False Alarm Rate (FAR) of each class (shown in Figures 5.19-5.23), we need to remember that having a low FAR is needed, but at the same time we need the high DR. So, FAR and DR always need to be interpreted together. Taking this into consideration, the low FAR achieved for R2L and U2R (in all cases) is useless as it comes with low DR. While the low FAR achieved for DoS and Probe



(a) Training Time



(b) Testing Time

Figure 5.12: Classifier Time Performance

(in all cases) is useful as they have high DRs.

Figures 5.24-5.28 and Figures 5.29-5.33 show the precision and F1 score for each class, respectively.

We have presented all the metrics considering our IDS as a multi-class classifier. However, if we want to consider our IDS as a binary classifier, then we will have

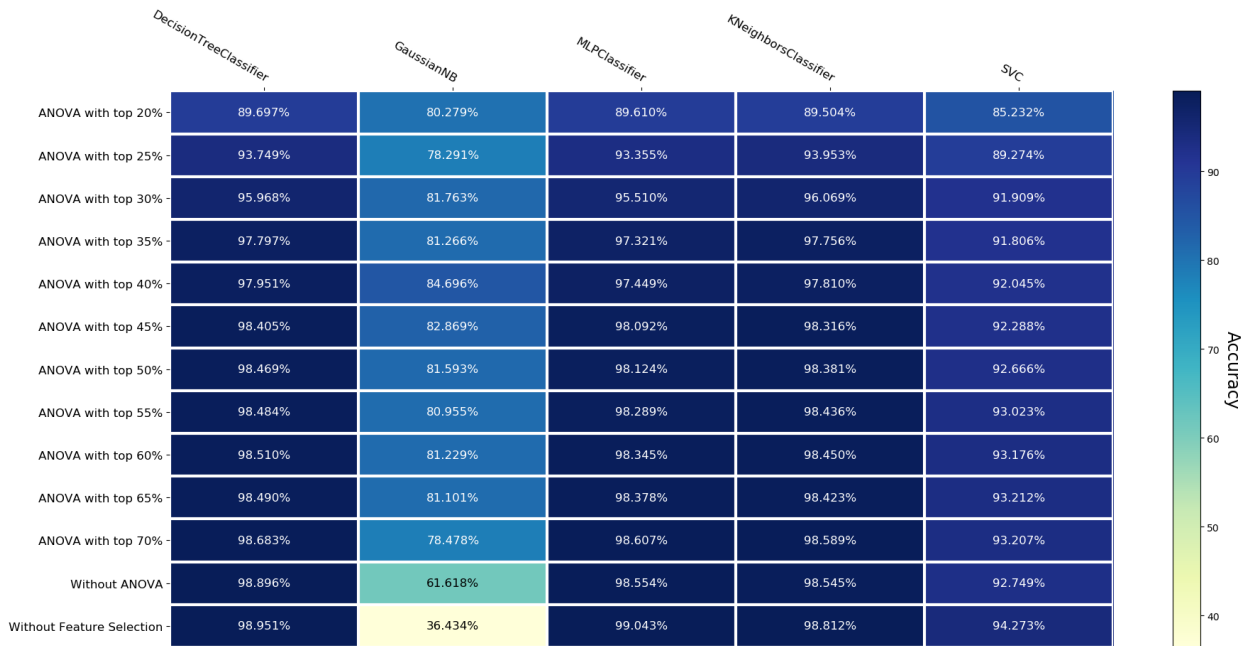


Figure 5.13: Classifier Overall Accuracy

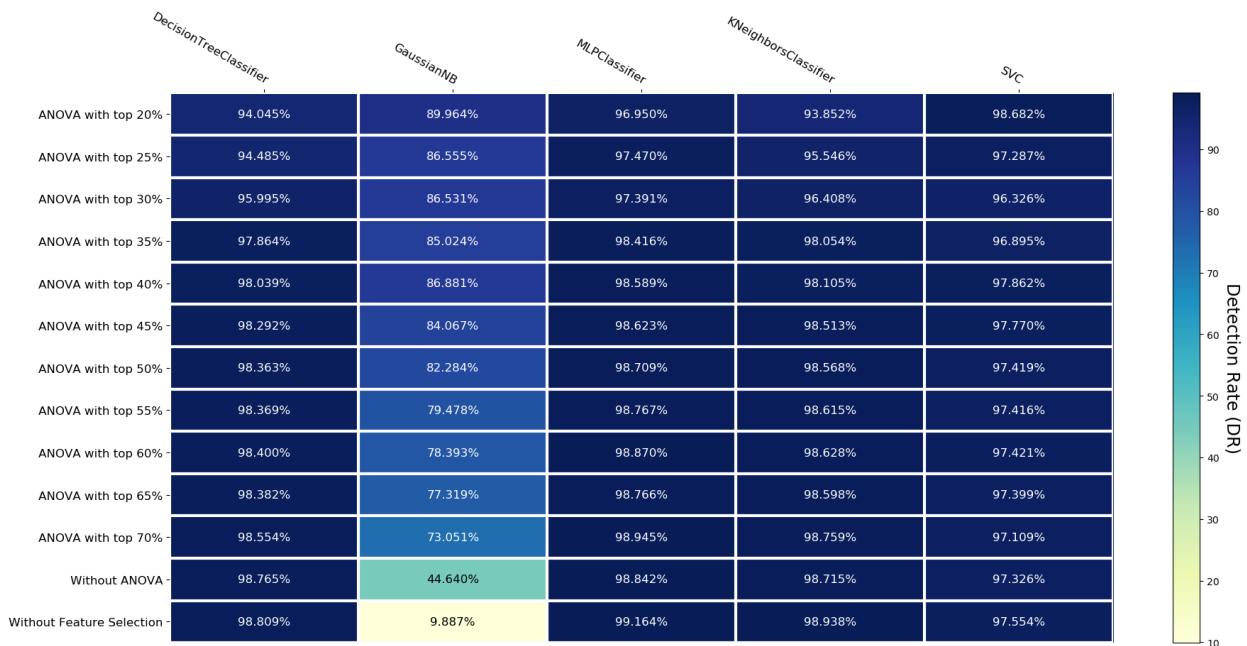


Figure 5.14: Normal Class Detection Rate

only two labels: normal and attack. Figures 5.34a and 5.34b show the training and testing time of the binary classifier. And Figures 5.35-5.37 show the accuracy, detection rate, and false alarm rate of the binary classifier.

It is noticed that using 50% as a threshold we have been able to achieve good time reduction in both training and testing for KNN and SVM, while achieving similar



Figure 5.15: DoS Class Detection Rate

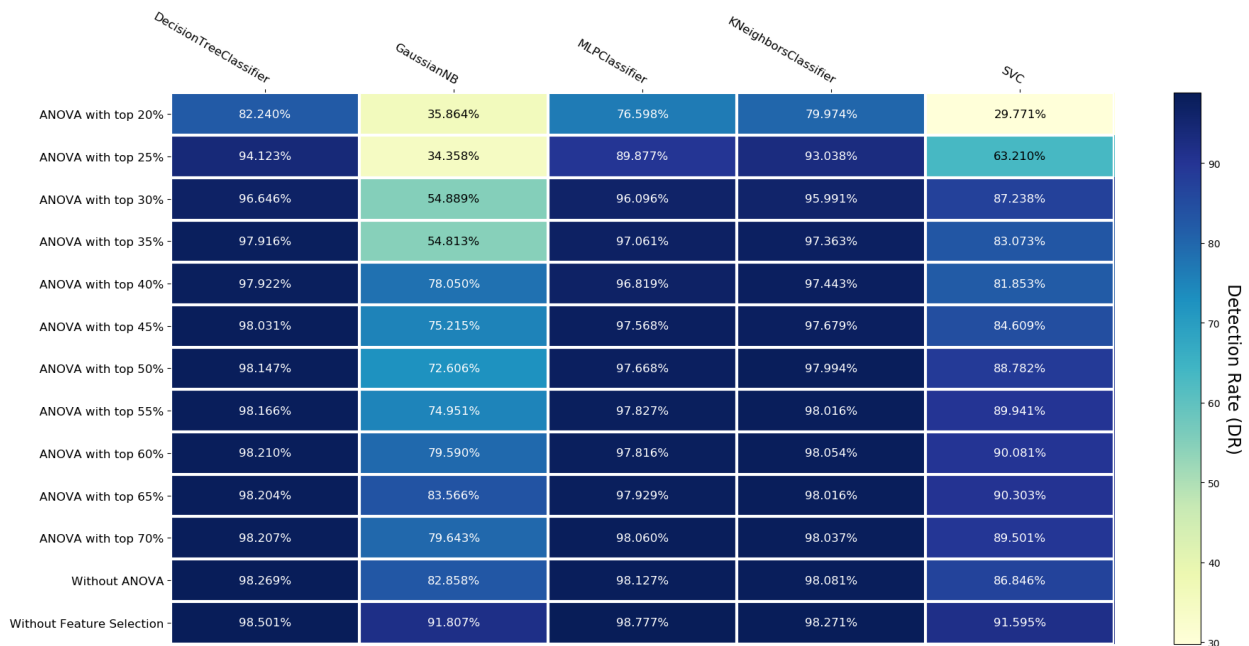


Figure 5.16: Probe Class Detection Rate



performance in the other metrics. For KNN, the training time is reduced from 17.264 seconds to 6.949 seconds (59% reduction), and the testing time is reduced from 33.803 seconds to 3.663 seconds (89% reduction). For SVM, the training time is reduced from 368.507 seconds to 119.024 seconds (68% reduction), and the testing time is reduced from 31.782 seconds to 16.565 seconds (48% reduction). As for the



Figure 5.17: R2L Class Detection Rate



Figure 5.18: U2R Class Detection Rate

other classifiers, they are already fast in training and testing, and we are achieving similar metrics.



Figure 5.19: Normal Class False Alarm Rate

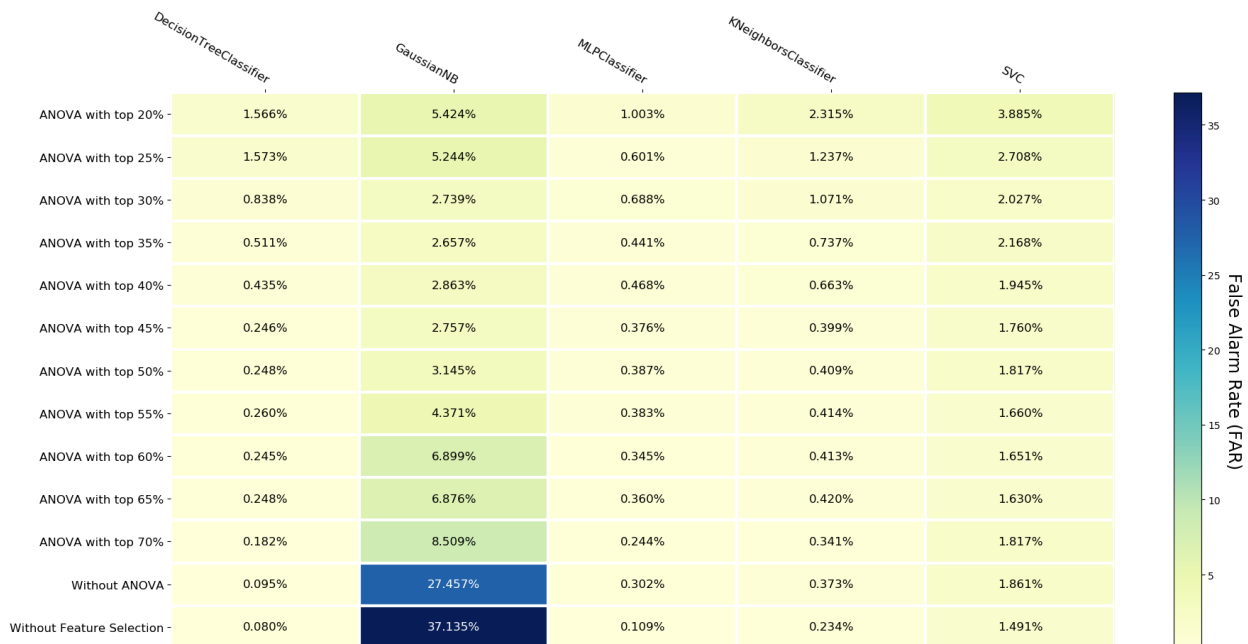


Figure 5.20: DoS Class False Alarm Rate



Figure 5.21: Probe Class False Alarm Rate



Figure 5.22: R2L Class False Alarm Rate



Figure 5.23: U2R Class False Alarm Rate

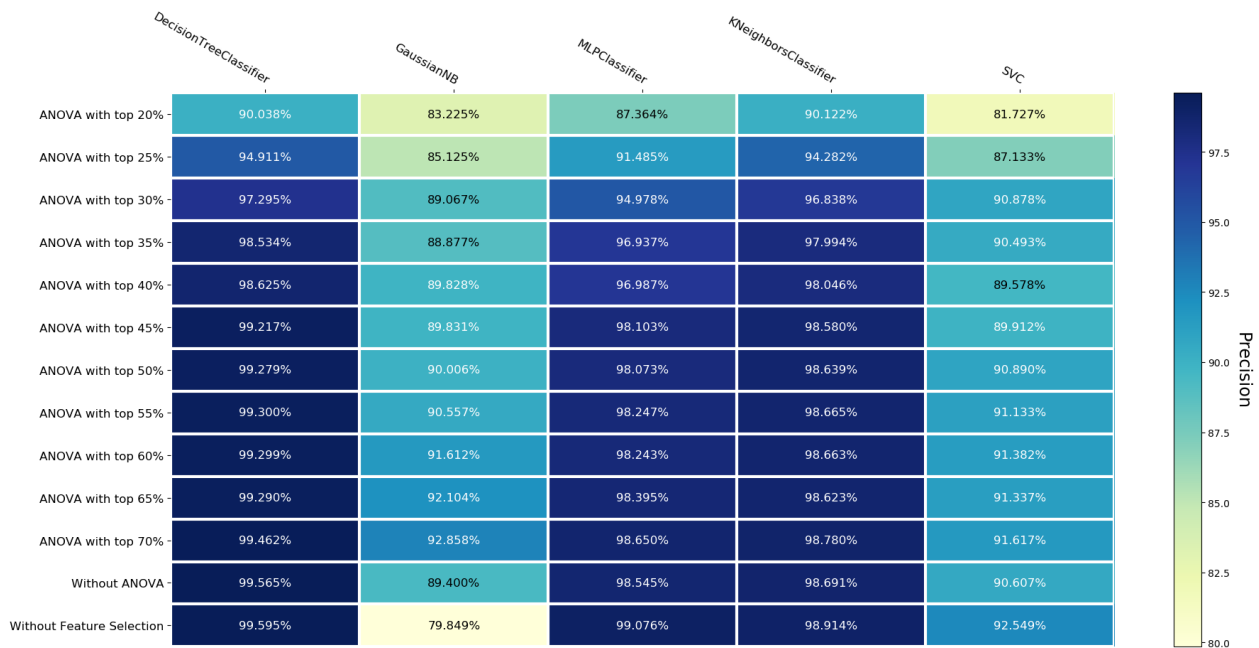


Figure 5.24: Normal Class Precision

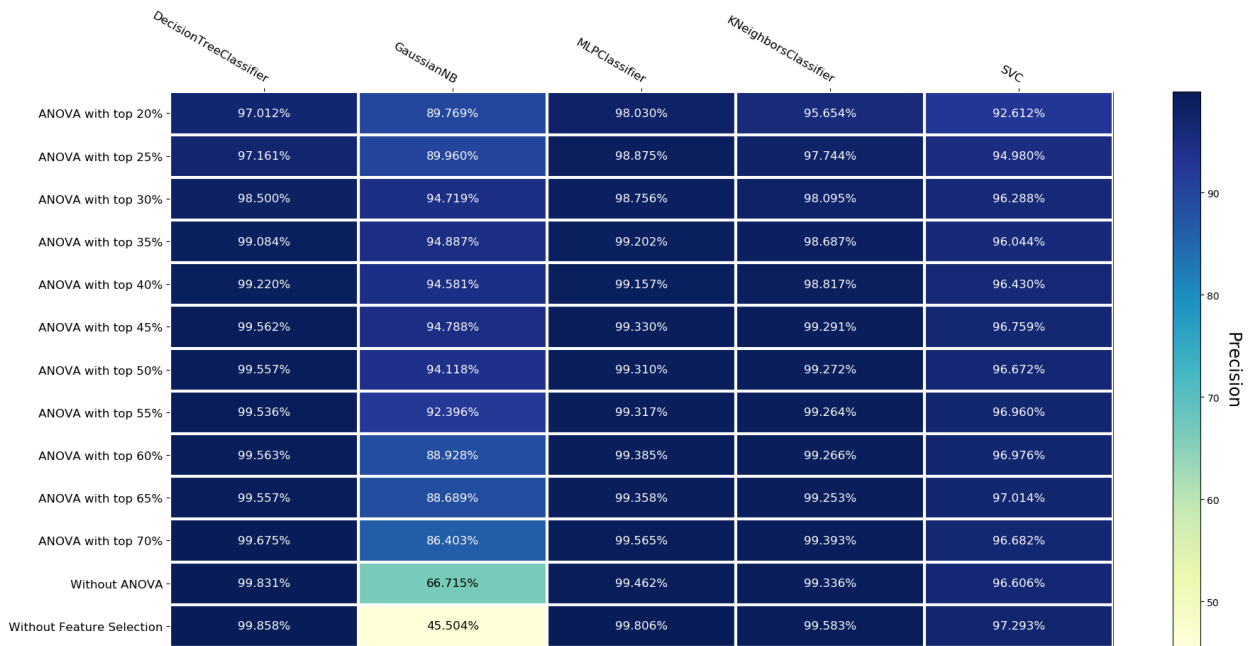


Figure 5.25: DoS Class Precision

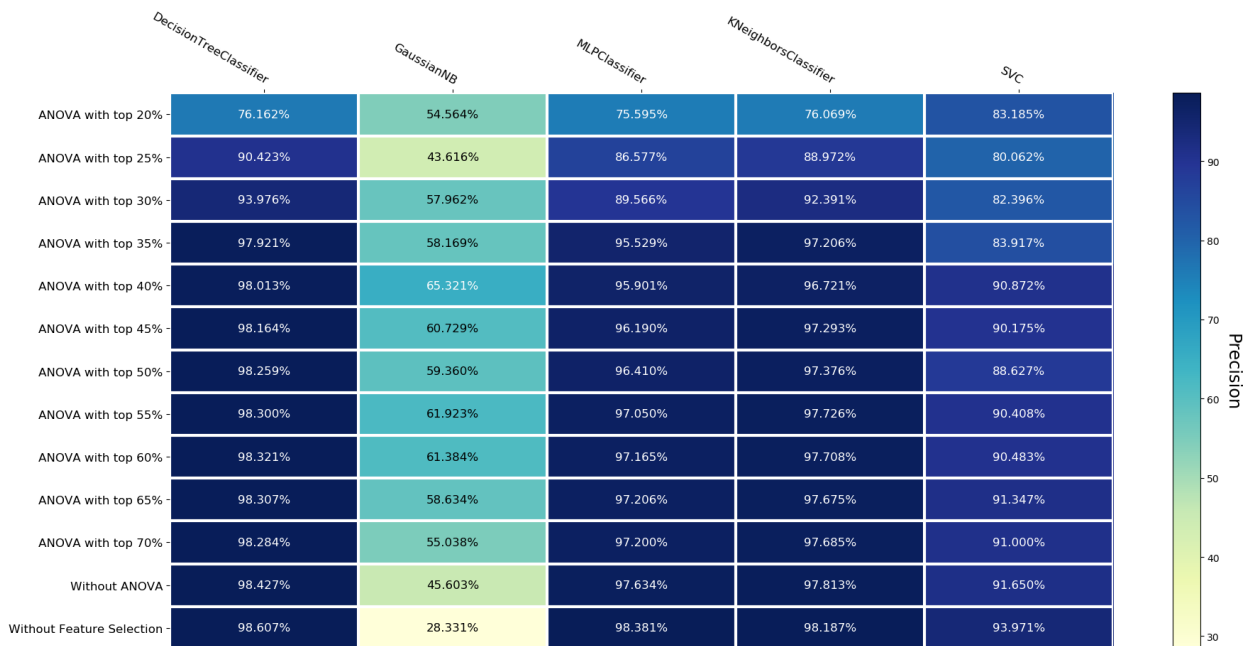


Figure 5.26: Probe Class Precision



Figure 5.27: R2L Class Precision



Figure 5.28: U2R Class Precision

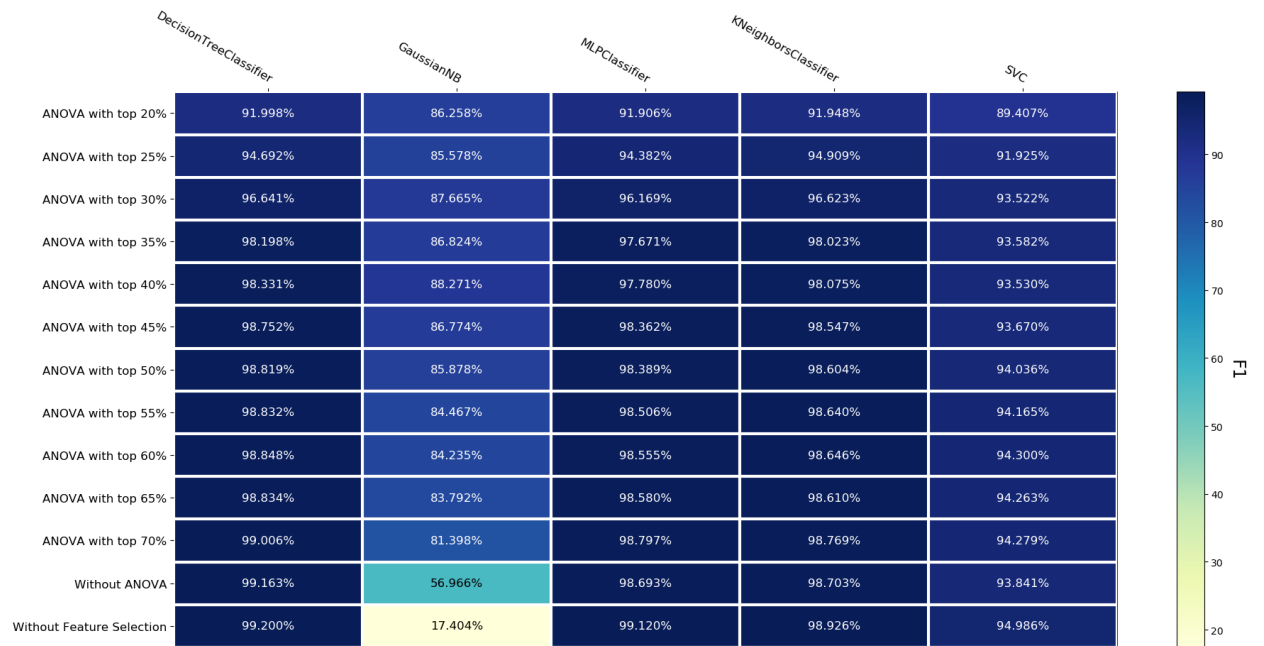


Figure 5.29: Normal Class F1 score

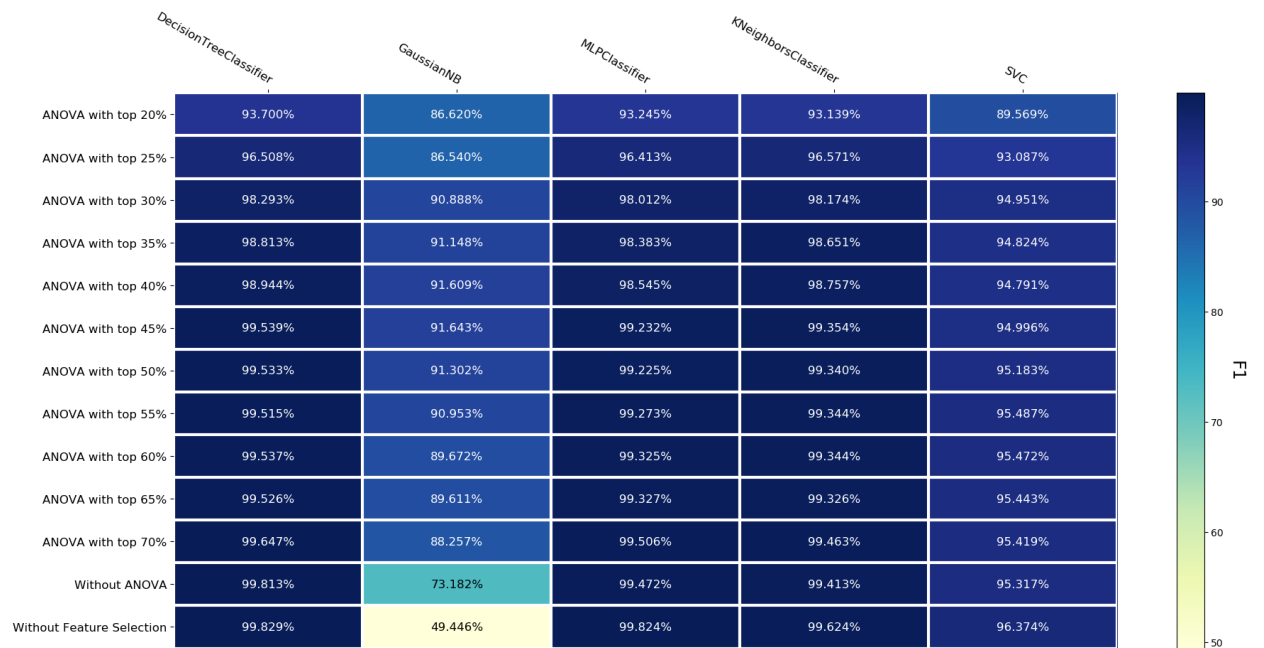


Figure 5.30: DoS Class F1 score

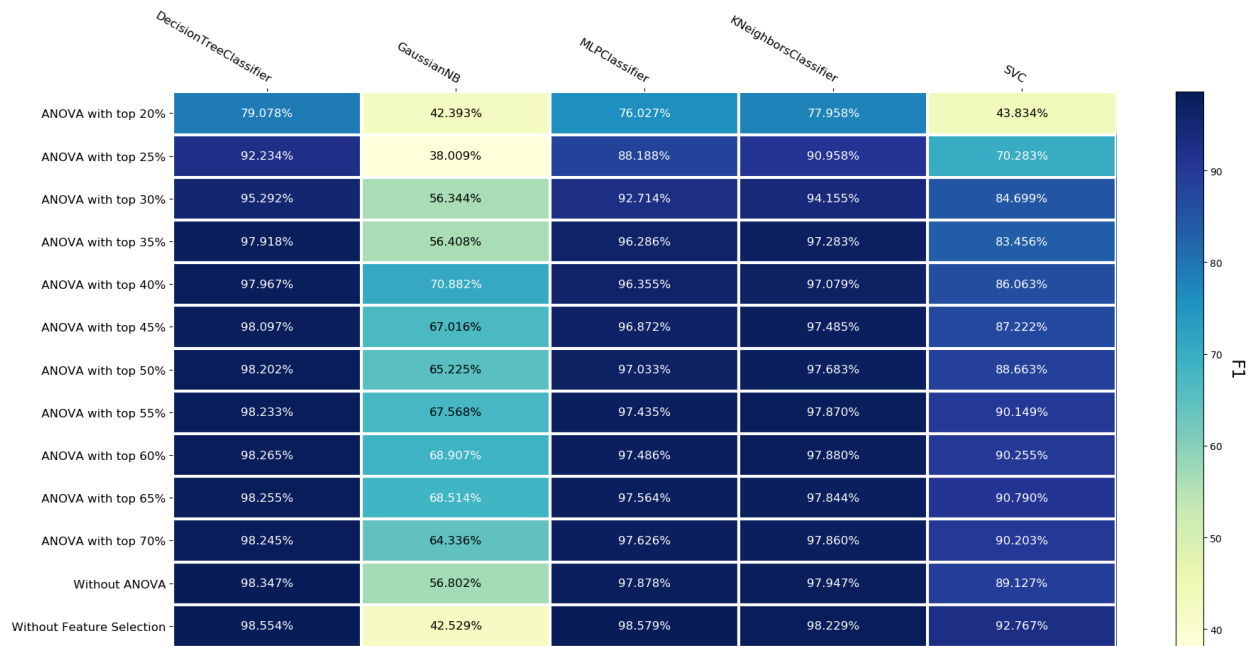


Figure 5.31: Probe Class F1 score

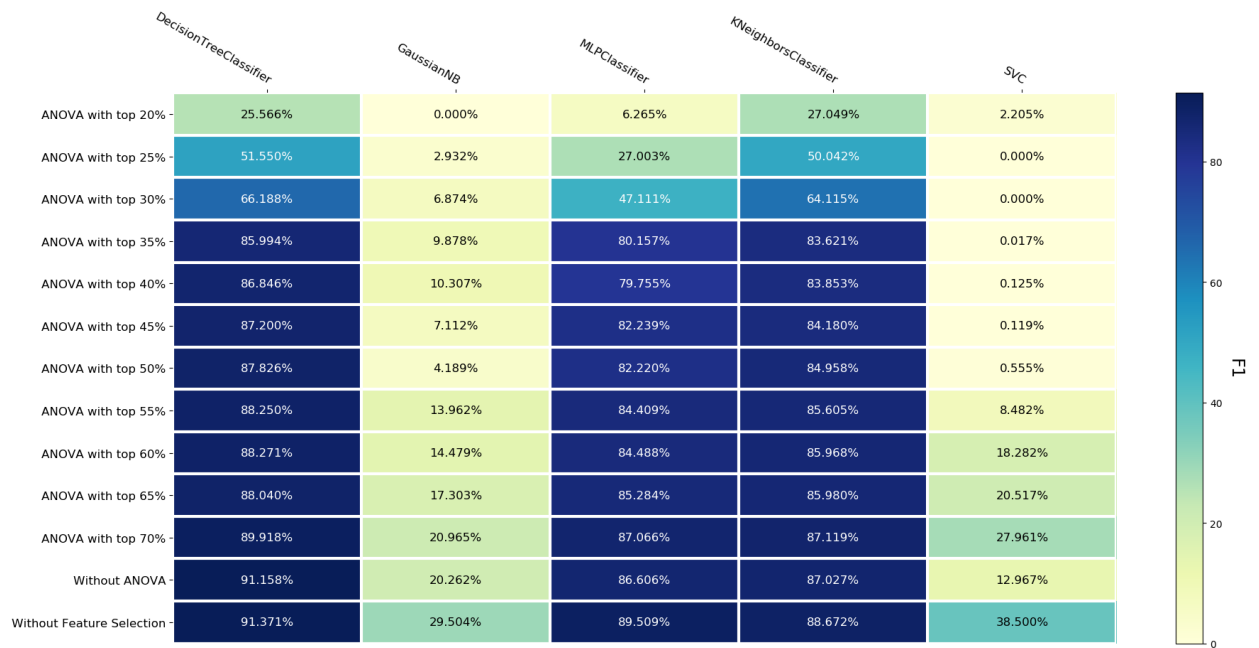


Figure 5.32: R2L Class F1 score

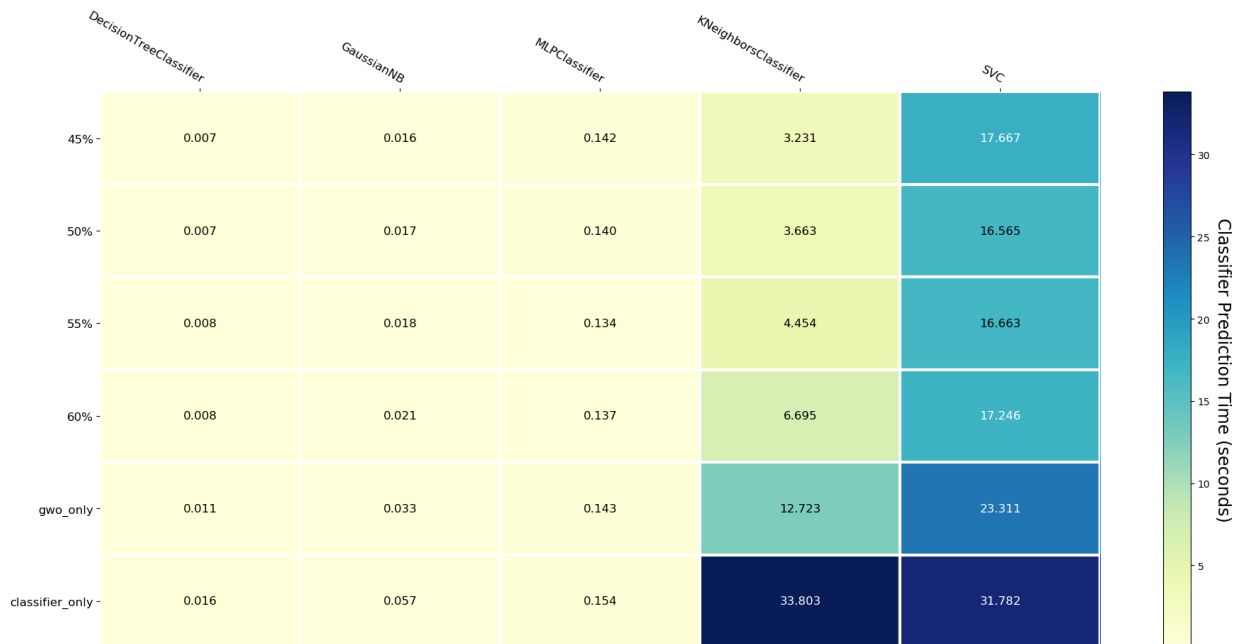




Figure 5.33: U2R Class F1 score



(a) Training Time



(b) Testing Time

Figure 5.34: Binary Classifier Time Performance

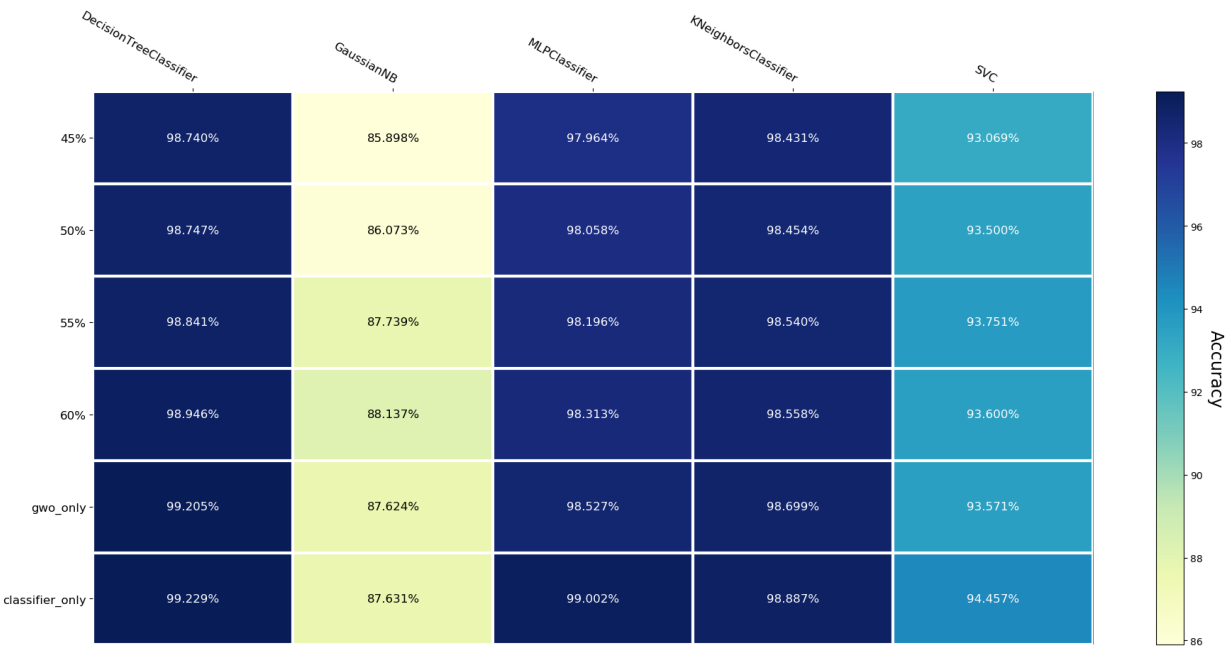


Figure 5.35: Binary Classifier Overall Accuracy

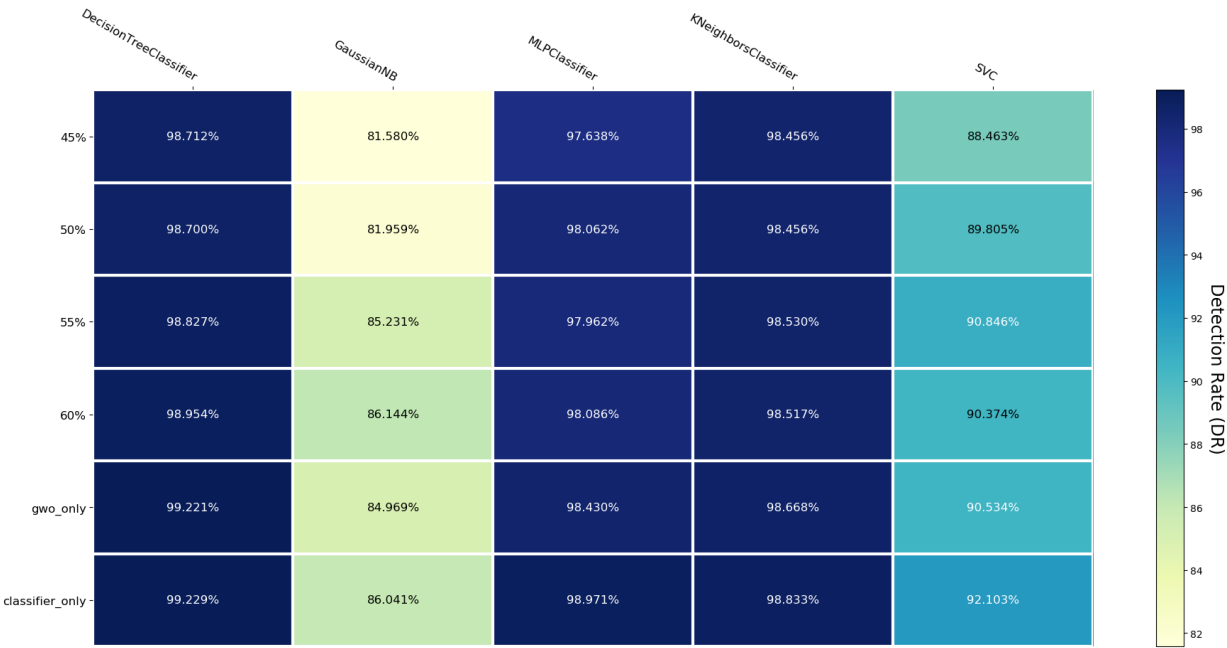


Figure 5.36: Binary Classifier Detection Rate



Figure 5.37: Binary Classifier False Alarm Rate



Figure 5.38: Binary Classifier Precision

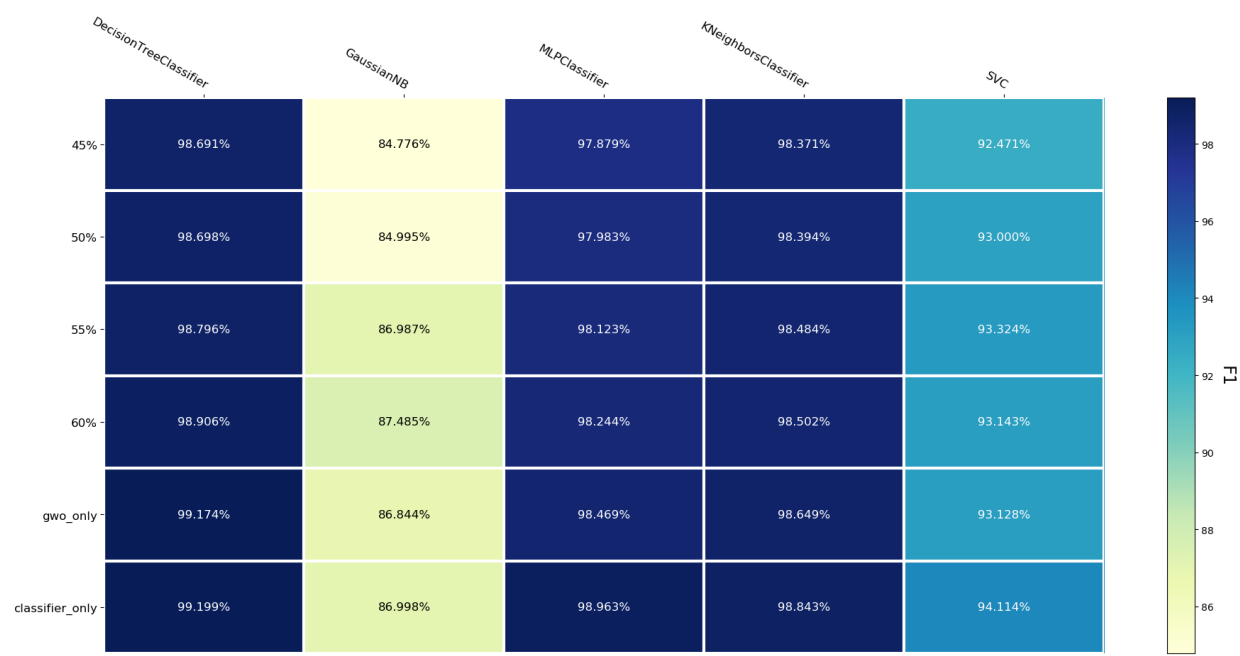


Figure 5.39: Binary Classifier F1 score

### 5.3 Summary

In this chapter, we discussed first how to tune GWO parameters. GWO performance is measured by its ability to converge fast and to the lowest possible fitness value. There are two main parameters that control GWO convergence and search time: population size, and maximum iterations. Population size is the number of solutions, and maximum iterations serves as the stopping criteria. After doing experiments on several combinations of these two parameters alongside with different ANOVA F-value thresholds, we have concluded that a maximum iterations of 10 is enough to converge, and a population size of 20 has the lowest starting value of fitness and the lowest ending value of fitness.

After that, we discussed thoroughly the performance of the proposed approach using the selected parameters for GWO. Throughout the experiments, we have concluded that the most balanced threshold for top ranked features from ANOVA F-value is a threshold of 50%. By balanced we mean, that it selects reasonable number of features that reduces the training time and testing time of the classifier while achieving similar classification performance.

In general, the proposed approach did not improve the classification performance of the classifiers, but at least it achieved similar results in less time and less dimension. The only exception to this is Naive Bayes, in which the overall accuracy was improved in multi-class classification. The major improvements in training time and testing time were in KNN and SVM, in both multi-class classification and binary classification.

# Chapter 6

## Conclusion and Future Work

In this study, we proposed a hybrid feature selection approach to address two challenges in any IDS:

1. Reduce dimensionality, and achieve short training time and near real-time prediction.
2. Achieve high detection rate, high accuracy, and low false alarm rate at the same time.

The hybrid approach uses ANOVA F-value in the first stage, and then it uses a wrapper method with GWO as a search strategy and Random Forest for evaluation in the second stage. We applied the proposed approach on NSL-KDD dataset, where we had first to transform nominal features to numerical using probability density function (PDF), and then we had to normalize numerical features using min-max. After that, we applied the proposed feature selection method. At the last stage, multi-class classification was applied using one-vs-rest strategy. We used five common classifiers: SVM, KNN, ANN, Naive Bayes, and Decision Tree.

We ran 30 experiments on the whole dataset. In each experiment, the data was randomly shuffled and then separated in a stratified fashion (to guarantee same distribution as original data) to 70% training data and 30% testing data. For each metric, the average of the 30 runs was computed.

The performance of the proposed approach was compared to the performance of classifiers without feature selection, and to feature selection using GWO only. After analyzing the results, we can conclude the following:

- The best parameters for having a well-performing GWO in terms of fast convergence and short search time are: a population size of 20 and maximum iterations of 10.
- The hybrid feature selection approach we proposed uses a threshold of 50% from feature ranking and then applies GWO. We have been able to reduce the search time of GWO by **18%** while selecting a smaller subset of features. Our approach resulted in a feature reduction from 41 to 13 (**68%** reduction), while GWO only reduced the features from 41 to 25 (39% reduction).
- Multi-class classification: we have aimed to reduce the classification training time and testing time, while achieving similar or better classification performance. However, we have seen that the proposed approach did not improve the classification performance except for few cases (e.g. GaussianNB classifier), but at least it achieved similar performance with better time and less dimension. For some classifiers, the reduction was negligible, while for others it was worthy. For KNN, the training time was reduced by **77%** and the testing time was reduced by **93%**. For SVM, the training time was reduced by **62%** and the testing time was reduced by **46%**.
- Binary classification: same applies here, we have been able to achieve similar performance with time reduction and less dimension. Again, the time reduction was obvious in KNN and SVM. For KNN, the training time was reduced by **59%** and the testing time was reduced by **89%**. For SVM, the training time was reduced by **68%** and the testing time was reduced by **48%**.

As noticed above, the proposed approach showed few improvements compared to the classifiers without feature selection. The justification behind this is that NSL-KDD is not that challenging to most computing processors nowadays. It has a dimension of 41 features, and it consists of around 150,000 records. The main reasons to choose it were its popularity, and its simplicity.

In the future, we aim to experiment the proposed approach on more recent and larger datasets, such as CIC-IDS-2017 and CSE-CIC-IDS-2018. We could not include those datasets in the scope of this study, because they need more analysis, data cleaning,



and data sampling. We also aim to experiment with several swarm intelligence algorithms such as FA and WSO.

# References

- [1] A. Thakkar and R. Lohiya, “A review of the advancement in intrusion detection datasets,” *Procedia Computer Science*, vol. 167, pp. 636–645, 2020.
- [2] T. Stevens, *Cyber Security and the Politics of Time*. Cambridge University Press, 2015.
- [3] S. M. H. Bamakan, H. Wang, T. Yingjie, and Y. Shi, “An effective intrusion detection framework based on mclp/svm optimized by time-varying chaos particle swarm optimization,” *Neurocomputing*, vol. 199, pp. 90–102, 2016.
- [4] Z.-H. Chen and C.-W. Tsai, “An effective metaheuristic algorithm for intrusion detection system,” in *2018 IEEE International Conference on Smart Internet of Things (SmartIoT)*. IEEE, 2018, pp. 154–159.
- [5] B. Selvakumar and K. Muneeswaran, “Firefly algorithm based feature selection for network intrusion detection,” *Computers & Security*, 2018.
- [6] V. Hajisalem and S. Babaie, “A hybrid intrusion detection system based on abc-afs algorithm for misuse and anomaly detection,” *Computer Networks*, vol. 136, pp. 37–50, 2018.
- [7] R. A. R. Ashfaq, X.-Z. Wang, J. Z. Huang, H. Abbas, and Y.-L. He, “Fuzziness based semi-supervised learning approach for intrusion detection system,” *Information Sciences*, vol. 378, pp. 484–497, 2017.
- [8] K. K. Vasan and B. Surendiran, “Dimensionality reduction using principal component analysis for network intrusion detection,” *Perspectives in Science*, vol. 8, pp. 510–512, 2016.

- [9] D. Papamartzivanos, F. G. Mármol, and G. Kambourakis, “Dendron: Genetic trees driven rule induction for network intrusion detection systems,” *Future Generation Computer Systems*, vol. 79, pp. 558–574, 2018.
- [10] M. Tavallaei, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the kdd cup 99 data set,” in *2009 IEEE symposium on computational intelligence for security and defense applications*. IEEE, 2009, pp. 1–6.
- [11] A. Shiravi, H. Shiravi, M. Tavallaei, and A. A. Ghorbani, “Toward developing a systematic approach to generate benchmark datasets for intrusion detection,” *computers & security*, vol. 31, no. 3, pp. 357–374, 2012.
- [12] I. Sharafaldin, A. H. Lashkari, and A. A. Ghorbani, “Toward generating a new intrusion detection dataset and intrusion traffic characterization.” in *ICISSP*, 2018, pp. 108–116.
- [13] Z. Zhao, F. Morstatter, S. Sharma, S. Alelyani, A. Anand, and H. Liu, “Advancing feature selection research,” *ASU feature selection repository*, pp. 1–28, 2010.
- [14] J. Tang, S. Alelyani, and H. Liu, “Feature selection for classification: A review,” *Data Classification: Algorithms and Applications*, p. 37, 2014.
- [15] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, “Feature selection: A data perspective,” *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–45, 2017.
- [16] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern classification*. John Wiley & Sons, 2012.
- [17] S. Wright, “The interpretation of population structure by f-statistics with special regard to systems of mating,” *Evolution*, pp. 395–420, 1965.
- [18] Y. Li, J.-L. Wang, Z.-H. Tian, T.-B. Lu, and C. Young, “Building lightweight intrusion detection system using wrapper-based feature selection mechanisms,” *Computers & Security*, vol. 28, no. 6, pp. 466–475, 2009.

- [19] H. Liu and R. Setiono, "Feature selection and classification-a probabilistic wrapper approach," in *Proceedings of 9th International Conference on Industrial and Engineering Applications of AI and ES*, 1997, pp. 419–424.
- [20] S. Mukherjee and N. Sharma, "Intrusion detection using naive bayes classifier with feature reduction," *Procedia Technology*, vol. 4, pp. 119–128, 2012.
- [21] H. ZHANG, R. TAO, Z.-y. LI, and Z.-h. CAI, "A research and application of feature selection based on knn and tabu search algorithm in the intrusion detection," *Acta Electronica Sinica*, vol. 7, 2009.
- [22] L. Davis, *Handbook of genetic algorithms*. Van Nostrand Reinhold, New York, 1991.
- [23] E. Emary, H. M. Zawbaa, and A. E. Hassanien, "Binary grey wolf optimization approaches for feature selection," *Neurocomputing*, vol. 172, pp. 371–381, 2016.
- [24] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *MHS'95. Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. Ieee, 1995, pp. 39–43.
- [25] M. Dorigo and L. M. Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on evolutionary computation*, vol. 1, no. 1, pp. 53–66, 1997.
- [26] X.-S. Yang, "Firefly algorithms for multimodal optimization," in *International symposium on stochastic algorithms*. Springer, 2009, pp. 169–178.
- [27] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey wolf optimizer," *Advances in engineering software*, vol. 69, pp. 46–61, 2014.
- [28] H. Faris, I. Aljarah, M. A. Al-Betar, and S. Mirjalili, "Grey wolf optimizer: a review of recent variants and applications," *Neural Computing and Applications*, pp. 1–23, 2017.
- [29] J. K. Seth and S. Chandra, "Intrusion detection based on key feature selection using binary gwo," in *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, 2016, pp. 3735–3740.

- [30] E. Devi and R. Suganthe, "Feature selection in intrusion detection grey wolf optimizer," *Asian Journal of Research in Social Sciences and Humanities*, vol. 7, no. 3, pp. 671–682, 2017.
- [31] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [32] N. Bambrick. (2016, Jul) Support vector machines: A simple explanation. Accessed: 2020-08-16. [Online]. Available: <https://www.kdnuggets.com/2016/07/support-vector-machines-simple-explanation.html>
- [33] A. Tandel. (2017, Aug) Support vector machines - a brief overview. Accessed: 2020-08-16. [Online]. Available: <https://towardsdatascience.com/support-vector-machines-a-brief-overview-37e018ae310f>
- [34] A. Kowalczyk. (2017, Apr) Svm - understanding the math - part 1 - the margin. Accessed: 2020-08-16. [Online]. Available: <https://www.svm-tutorial.com/2014/11/svm-understanding-math-part-1/>
- [35] B. V. Dasarathy, *Nearest Neighbor (NN) Norms NN pattern Classification Techniques*. IEEE Computer Society Press, 01 1991.
- [36] J. Han, J. Pei, and M. Kamber, *Data mining: concepts and techniques*. Elsevier, 2011.
- [37] N. F. Haq, A. R. Onik, M. A. K. Hridoy, M. Rafni, F. M. Shah, and D. M. Farid, "Application of machine learning approaches in intrusion detection system: a survey," *IJARAI-International Journal of Advanced Research in Artificial Intelligence*, vol. 4, no. 3, pp. 9–18, 2015.
- [38] I. T. Jolliffe, "Principal components in regression analysis," in *Principal component analysis*. Springer, 1986, pp. 129–155.
- [39] P. Comon, "Independent component analysis, a new concept?" *Signal processing*, vol. 36, no. 3, pp. 287–314, 1994.
- [40] S. Balakrishnama and A. Ganapathiraju, "Linear discriminant analysis-a brief tutorial," in *Institute for Signal and information Processing*, vol. 18, no. 1998, 1998, pp. 1–8.

- [41] S. T. Roweis and L. K. Saul, “Nonlinear dimensionality reduction by locally linear embedding,” *science*, vol. 290, no. 5500, pp. 2323–2326, 2000.
- [42] A. Almomani, M. Alweshah, and S. Al, “Metaheuristic algorithms-based feature selection approach for intrusion detection,” *Machine Learning for Computer and Cyber Security: Principle, Algorithms, and Practices*, p. 184, 2019.
- [43] K. Kira and L. A. Rendell, “A practical approach to feature selection,” in *Machine Learning Proceedings 1992*. Elsevier, 1992, pp. 249–256.
- [44] H. Liu and R. Setiono, “Chi2: Feature selection and discretization of numeric attributes,” in *Proceedings of 7th IEEE International Conference on Tools with Artificial Intelligence*. IEEE, 1995, pp. 388–391.
- [45] C. Gini, “Variability and mutability, contribution to the study of statistical distribution and relaitons,” *Studi Economico-Giuricici della R*, 1912.
- [46] R. Tibshirani, “Regression shrinkage and selection via the lasso,” *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 58, no. 1, pp. 267–288, 1996.
- [47] F. Glover and M. Laguna, “General purpose heuristics for integer programming—part i,” *Journal of Heuristics*, vol. 2, no. 4, pp. 343–358, 1997.
- [48] C. Blum and A. Roli, “Metaheuristics in combinatorial optimization: Overview and conceptual comparison,” *ACM computing surveys (CSUR)*, vol. 35, no. 3, pp. 268–308, 2003.
- [49] S. Voß, “Meta-heuristics: The state of the art,” in *Local Search for Planning and Scheduling*, A. Nareyek, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 1–23.
- [50] P. J. Van Laarhoven and E. H. Aarts, “Simulated annealing,” in *Simulated annealing: Theory and applications*. Springer, 1987, pp. 7–15.
- [51] F. Glover, “Tabu search—part i,” *ORSA Journal on computing*, vol. 1, no. 3, pp. 190–206, 1989.

- [52] H. Faris, I. Aljarah, M. A. Al-Betar, and S. Mirjalili, “Grey wolf optimizer: a review of recent variants and applications,” *Neural computing and applications*, pp. 1–23, 2018.
- [53] R. Storn and K. Price, “Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces,” *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.
- [54] D. Karaboga, “An idea based on honey bee swarm for numerical optimization,” Technical report-tr06, Erciyes university, engineering faculty, computer ..., Tech. Rep., 2005.
- [55] R. K. Cunningham, R. P. Lippmann, D. J. Fried, S. L. Garfinkel, I. Graf, K. R. Kendall, S. E. Webster, D. Wyschogrod, and M. A. Zissman, “Evaluating intrusion detection systems without attacking your friends: The 1998 darpa intrusion detection evaluation,” MASSACHUSETTS INST OF TECH LEXINGTON LINCOLN LAB, Tech. Rep., 1999.
- [56] Kdd cup 1999. Accessed: 2020-03-03. [Online]. Available: <http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html>
- [57] J. Song, H. Takakura, Y. Okabe, M. Eto, D. Inoue, and K. Nakao, “Statistical analysis of honeypot data and building of kyoto 2006+ dataset for nids evaluation,” in *Proceedings of the first workshop on building analysis datasets and gathering experience returns for security*, 2011, pp. 29–36.
- [58] E. Hodo, X. J. A. Bellekens, A. W. Hamilton, C. Tachtatzis, and R. C. Atkinson, “Shallow and deep networks intrusion detection system: A taxonomy and survey,” *CoRR*, vol. abs/1701.02145, 2017. [Online]. Available: <http://arxiv.org/abs/1701.02145>
- [59] H. Hindy, D. Brosset, E. Bayne, A. Seeam, C. Tachtatzis, R. C. Atkinson, and X. J. A. Bellekens, “A taxonomy and survey of intrusion detection system design techniques, network threats and datasets,” *CoRR*, vol. abs/1806.03517, 2018. [Online]. Available: <http://arxiv.org/abs/1806.03517>

- [60] F. Salo, M. Injadat, A. B. Nassif, A. Shami, and A. Essex, “Data mining techniques in intrusion detection systems: A systematic literature review,” *IEEE Access*, vol. 6, pp. 56 046–56 058, 2018.
- [61] M. M. Lisehroodi, Z. Muda, and W. Yassin, “A hybrid framework based on neural network mlp and k-means clustering for intrusion detection system,” in *4th International Conference on Computing and Informatics, ICOCI*, 2013.
- [62] R. Ranjan and G. Sahoo, “A new clustering approach for anomaly intrusion detection,” *CoRR*, vol. abs/1404.2772, 2014. [Online]. Available: <http://arxiv.org/abs/1404.2772>
- [63] S. Lee, G. Kim, and S. Kim, “Self-adaptive and dynamic clustering for online anomaly detection,” *Expert Systems with Applications*, vol. 38, no. 12, pp. 14 891–14 898, 2011.
- [64] A. Saied, R. E. Overill, and T. Radzik, “Detection of known and unknown ddos attacks using artificial neural networks,” *Neurocomputing*, vol. 172, pp. 385–393, 2016.
- [65] C. Yin, Y. Zhu, J. Fei, and X. He, “A deep learning approach for intrusion detection using recurrent neural networks,” *IEEE Access*, vol. 5, pp. 21 954–21 961, 2017.
- [66] M. Govindarajan and R. Chandrasekaran, “Intrusion detection using neural based hybrid classification methods,” *Computer networks*, vol. 55, no. 8, pp. 1662–1671, 2011.
- [67] Y. Yi, J. Wu, and W. Xu, “Incremental svm based on reserved set for network intrusion detection,” *Expert Systems with Applications*, vol. 38, no. 6, pp. 7698–7707, 2011.
- [68] L. Koc, T. A. Mazzuchi, and S. Sarkani, “A network intrusion detection system based on a hidden naïve bayes multiclass classifier,” *Expert Systems with Applications*, vol. 39, no. 18, pp. 13 492–13 500, 2012.



- [69] S. Mohammadi, H. Mirvaziri, M. Ghazizadeh-Ahsaei, and H. Karimipour, "Cyber intrusion detection by combined feature selection algorithm," *Journal of information security and applications*, vol. 44, pp. 80–88, 2019.
- [70] B. Hajimirzaei and N. J. Navimipour, "Intrusion detection for cloud computing using neural networks and artificial bee colony optimization algorithm," *ICT Express*, vol. 5, no. 1, pp. 56–59, 2019.
- [71] A. A. Aburomman and M. B. I. Reaz, "A novel svm-knn-pso ensemble method for intrusion detection system," *Applied Soft Computing*, vol. 38, pp. 360–372, 2016.
- [72] N. Moustafa, J. Slay, and G. Creech, "Novel geometric area analysis technique for anomaly detection using trapezoidal area estimation on large-scale networks," *IEEE Transactions on Big Data*, 2017.
- [73] T. Hamed, J. B. Ernst, and S. C. Kremer, "A survey and taxonomy of classifiers of intrusion detection systems," in *Computer and network security essentials*. Springer, 2018, pp. 21–39.
- [74] M. H. Bhuyan, D. K. Bhattacharyya, and J. K. Kalita, "Network anomaly detection: methods, systems and tools," *Ieee communications surveys & tutorials*, vol. 16, no. 1, pp. 303–336, 2013.
- [75] Z. D. Yenice, N. Adhikari, Y. K. Wong, V. Aksakalli, A. Taskin Gumus, and B. Abbasi, "SPSA-FSR: Simultaneous Perturbation Stochastic Approximation for Feature Selection and Ranking," *arXiv e-prints*, p. arXiv:1804.05589, Apr. 2018.
- [76] B. C. Beins and M. A. McCarthy, *Research methods and statistics*. Cambridge University Press, 2017.
- [77] Understanding analysis of variance (anova) and the f-test. Accessed: 2020-07-30. [Online]. Available: <https://blog.minitab.com/blog/adventures-in-statistics-2/understanding-analysis-of-variance-anova-and-the-f-test>

- [78] H. J. Escalante, S. V. Rodriguez, J. Cordero, A. R. Kristensen, and C. Cornou, “Sow-activity classification from acceleration patterns: a machine learning approach,” *Computers and electronics in agriculture*, vol. 93, pp. 17–26, 2013.
- [79] S. K. Biswas, “Intrusion detection using machine learning: A comparison study,” *International Journal of Pure and Applied Mathematics*, vol. 118, no. 19, pp. 101–114, 2018.
- [80] Numpy library. Accessed: 2020-08-13. [Online]. Available: <https://numpy.org/>
- [81] pandas library. Accessed: 2020-08-13. [Online]. Available: <https://pandas.pydata.org/>
- [82] Matplotlib library. Accessed: 2020-08-13. [Online]. Available: <https://matplotlib.org/>
- [83] H. Faris, I. Aljarah, S. Mirjalili, P. A. Castillo, and J. J. Merelo, “Evolopy: An open-source nature-inspired optimization framework in python.” in *IJCCI (ECTA)*, 2016, pp. 171–177.
- [84] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, “Scikit-learn: Machine learning in python,” *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

# Appendix A

## Implementation

```
[ ]: import feather
import numpy as np
from sklearn.model_selection import train_test_split
```

First we define the features order as in NSL-KDD

```
[ ]: basic_features = ["duration",
                      "protocol_type",
                      "service",
                      "flag",
                      "src_bytes",
                      "dst_bytes",
                      "land",
                      "wrong_fragment",
                      "urgent"]

content_features = ["hot",
                   "num_failed_logins",
                   "logged_in",
                   "num_compromised",
                   "root_shell",
                   "su_attempted",
                   "num_root",
```

```
        "num_file_creations",
        "num_shells",
        "num_access_files",
        "num_outbound_cmds",
        "is_host_login",
        "is_guest_login"]

time_based_traffic_features = ["count",
                               "srv_count",
                               "serror_rate",
                               "srv_serror_rate",
                               "rerror_rate",
                               "srv_rerror_rate",
                               "same_srv_rate",
                               "diff_srv_rate",
                               "srv_diff_host_rate"]

host_based_traffic_features = ["dst_host_count",
                               "dst_host_srv_count",
                               "dst_host_same_srv_rate",
                               "dst_host_diff_srv_rate",
                               "dst_host_same_src_port_rate",
                               "dst_host_srv_diff_host_rate",
                               "dst_host_serror_rate",
                               "dst_host_srv_serror_rate",
                               "dst_host_rerror_rate",
                               "dst_host_srv_rerror_rate"]

attr_names = basic_features + content_features + \
    time_based_traffic_features + host_based_traffic_features
col_names = attr_names + ["label", "difficulty"]
```

Then, we define the mapping of NSL-KDD to five categories

```
[ ]: # Normal --> 0, DoS --> 1, Probe --> 2, R2L --> 3, U2R --> 4
five_labels_dict = {'normal': 0, 'neptune': 1, 'back': 1, 'land': 1,
                    'pod': 1, 'smurf': 1, 'teardrop': 1,
                    'mailbomb': 1,
                    'apache2': 1,
                    'processtable': 1, 'udpstorm': 1, 'worm': 1,
                    'ipsweep': 2, 'nmap': 2, 'portsweep': 2,
                    'satan': 2, 'mscan': 2, 'saint': 2,
                    'ftp_write': 3, 'guess_passwd': 3, 'imap': 3,
                    'multihop': 3, 'phf': 3, 'spy': 3,
                    'warezclient': 3,
                    'warezmaster': 3, 'sendmail': 3, 'named': 3,
                    'snmpgetattack': 3, 'snmpguess': 3, 'xlock': 3,
                    'xsnoop': 3,
                    'httptunnel': 3,
                    'buffer_overflow': 4, 'loadmodule': 4, 'perl': 4,
                    'rootkit': 4, 'ps': 4, 'sqlattack': 4,
                    'xterm': 4}
```

Here we define the function that will read NSL-KDD, shuffle the data and split it to 70% training and 30% testing

```
[ ]: def read_data(multi_class, rand, test_size=0.3):
    df = feather.read_dataframe("KDDTrain+.feather")
    df_test = feather.read_dataframe("KDDTest+.feather")
    df.drop("difficulty", axis=1, inplace=True)
    df_test.drop("difficulty", axis=1, inplace=True)

    X_train_original = df.drop('label', axis=1)
    X_test_original = df_test.drop('label', axis=1)
    X_final = X_train_original.append(X_test_original)
```

```

y_train_original = df['label'].replace(five_labels_dict).
↪values.astype('int').ravel()

y_test_original = df_test['label'].replace(five_labels_dict).
↪values.astype('int').ravel()

y_final = np.concatenate((y_train_original, y_test_original))

X_train, X_test, y_train, y_test = train_test_split(X_final,
↪y_final, test_size=test_size, random_state=rand,
↪stratify=y_final)

if not multi_class:
    y_train[y_train > 0] = 1
    y_test[y_test > 0] = 1

return X_train, X_test, y_train, y_test

```

This is the implementation of PDF used for transforming nominal features to numerical

```

[ ]: from sklearn.base import BaseEstimator, TransformerMixin

def column_pdf(df, column_name):
    column_histogram = df[column_name].value_counts()
    column_sum = column_histogram.sum()
    return column_histogram.transform(lambda s: s / column_sum)

class PdfTransformer(BaseEstimator, TransformerMixin):

    def __init__(self):
        super().__init__()

```

```
def fit(self, X, y=None):
    self.columnns_pdfs = {clmn: column_pdf(X, clmn).to_dict()
    ↪for clmn, _ in X.iteritems()}
    return self

def transform(self, X, y='', copy=None):
    for clmn, _ in X.iteritems():
        X[clmn] = X[clmn].map(self.columnns_pdfs[clmn]).
    ↪fillna(0)
    return X
```

```
[ ]: import random
import time

from joblib import Parallel, delayed
from skfuzzy.membership import sigmf
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

This class will contain all the attributes we want from GWO

```
[ ]: class solution:
    def __init__(self):
        self.best = 0
        self.bestIndividual = []
        self.convergence = []
        self.features = []
        self.optimizer = ""
        self.objfname = ""
        self.startTime = 0
        self.endTime = 0
        self.executionTime = 0
```

```
self.lb = 0
self.ub = 0
self.dim = 0
self.popnum = 0
self.maxiers = 0
```

This is the cross over used in GWO

```
[ ]: def cross_over(x_1, x_2, x_3):
    r = random.random()
    if r < 0.333:
        return x_1
    elif r < 0.6666:
        return x_2
    else:
        return x_3
```

And this is the fitness function used in GWO

```
[ ]: def fitness_func(args):
    positions, X_train, y_train, X_test, y_test = args

    sz_w = 0.01

    """
    Fitness function to be passed to bGWO
    :param x: current positions for a specific search agent in
    ↪ bGWO
    :return: fitness value
    """
    current_selected_features_indexes = np.where(positions == 1)[0]
    X_train_current = np.copy(X_train[:,
    ↪ current_selected_features_indexes[
```



```
                                0:])) # slice training
↪data based on current_selected_features_indexes
    X_test_current = np.copy(X_test[:,
↪current_selected_features_indexes[
                                0:])) # slice testing data
↪based on current_selected_features_indexes
    # a case where no features were selected, the fitness will
↪be inf, indicating a bad classification
    if np.sum(positions) == 0:
        return float("inf")
    bgwo_fitness_classifier = RandomForestClassifier(n_jobs=-1,
↪n_estimators=5)
    bgwo_fitness_classifier = bgwo_fitness_classifier.
↪fit(X_train_current, y_train)
    # predict the class labels of test data
    y_predict = bgwo_fitness_classifier.predict(X_test_current)
    success_rate = accuracy_score(y_test, y_predict)
    return (1 - sz_w) * (1 - success_rate) + sz_w * np.
↪sum(positions) / np.alen(positions)
```

And this the implementation of BGWO (inspired from the original implementation in Matlab and from GWO implementation in EvoloPy repository)

```
[ ]: def BGWO(X, y, search_agents_no, max_iter, test_size=0.30,
↪random_state=42):
    dim = X.shape[1]

    X_train, X_test, y_train, y_test = train_test_split(X, y,
↪test_size=test_size, random_state=random_state)

    lb = 0
```

```

ub = 1
mb = float((lb + ub) / 2)

# initialize alpha, beta, and delta_pos
alpha_pos = np.zeros(dim)
alpha_score = float("inf")

beta_pos = np.zeros(dim)
beta_score = float("inf")

delta_pos = np.zeros(dim)
delta_score = float("inf")

# Initialize the positions of search agents
positions = np.random.uniform(lb, ub, (search_agents_no, dim))
↪ * (ub - lb) + lb

# make sure that all positions are 0 and 1
positions = positions > 0.5
positions = positions.astype(int)

convergence_curve = np.zeros(max_iter)
features_curve = np.zeros(max_iter)
s = solution()

timer_start = time.time()
s.startTime = time.strftime("%Y-%m-%d-%H-%M-%S")

# Main loop
for l in range(0, max_iter):
    nargs = [(positions[i, :].copy(), X_train, y_train,
↪ X_test, y_test) for i in range(0, search_agents_no)]

```

```
fitness_list = Parallel(n_jobs=-1,
↳prefer='threads')(delayed(fitness_func)(arg) for arg in nargs)

for i in range(0, search_agents_no):
    fitness = fitness_list[i]
    # Update Alpha, Beta, and Delta
    if fitness < alpha_score:
        alpha_score = fitness # Update alpha
        alpha_pos = positions[i, :].copy()
        s.bestIndividual = alpha_pos # always store best
↳solution so far

    if alpha_score < fitness < beta_score:
        beta_score = fitness # Update beta
        beta_pos = positions[i, :].copy()

    if alpha_score < fitness < delta_score and fitness >
↳beta_score:
        delta_score = fitness # Update delta
        delta_pos = positions[i, :].copy()

    a = 2 - 1 * (2 / max_iter) # a decreases linearly from 2
↳to 0

    # Update the Position of search agents including omegas
    for i in range(0, search_agents_no):
        for j in range(0, dim):
            r1 = random.random() # r1 is a random number in
↳[0,1]

            r2 = random.random() # r2 is a random number in
↳[0,1]
```

```

a1 = 2 * a * r1 - a # Equation (3.3)
c1 = 2 * r2 # Equation (3.4)

d_alpha = abs(c1 * alpha_pos[j] - positions[i, j])
↪ # Equation (3.5)-part 1

v1 = sigmf(-a1 * d_alpha, 0.5, 10) # apply
↪ sigmoid on -a1 * d_alpha

v1 = 0 if v1 < np.random.rand() else 1
x1 = alpha_pos[j] + v1 # Equation (3.6)-part 1
x1 = int(x1 >= 1)

r1 = random.random() # r1 is a random number in
↪ [0,1]

r2 = random.random() # r2 is a random number in
↪ [0,1]

a2 = 2 * a * r1 - a # Equation (3.3)
c2 = 2 * r2 # Equation (3.4)

d_beta = abs(c2 * beta_pos[j] - positions[i, j])
↪ # Equation (3.5)-part 2

v1 = sigmf(-a2 * d_beta, 0.5, 10) # apply sigmoid
↪ on -a2 * d_beta

v1 = 0 if v1 < np.random.rand() else 1
x2 = beta_pos[j] + v1 # Equation (3.6)-part 2
x2 = int(x2 >= 1)

r1 = random.random()
r2 = random.random()

a3 = 2 * a * r1 - a # Equation (3.3)

```

```
        c3 = 2 * r2 # Equation (3.4)

        d_delta = abs(c3 * delta_pos[j] - positions[i, j])
↪ # Equation (3.5)-part 3

        v1 = sigmf(-a3 * d_delta, 0.5, 10) # apply
↪ sigmoid on -a3 * d_delta

        v1 = 0 if v1 < np.random.rand() else 1
        x3 = delta_pos[j] + v1 # Equation (3.6)-part 2
        x3 = int(x3 >= 1)

        positions[i, j] = cross_over(x1, x2, x3)

    convergence_curve[1] = alpha_score
    features_curve[1] = np.sum(alpha_pos)

    timer_end = time.time()
    s.endTime = time.strftime("%Y-%m-%d-%H-%M-%S")
    s.executionTime = timer_end - timer_start
    s.convergence = convergence_curve
    s.features = features_curve
    s.optimizer = "GWO"
    s.objfname = fitness_func.__name__

    return s
```

We define BGWO as a classifier following sklearn pattern

```
[ ]: from sklearn.feature_selection.base import SelectorMixin

class BgwoClassifier(BaseEstimator, SelectorMixin):

    def __init__(self, population_size, iterations, test_size,
↪ random_state):
```

```

        self.population_size = population_size
        self.iterations = iterations
        self.test_size = test_size
        self.random_state = random_state

    def fit(self, X, y):
        # solution = WBGWO(X, y, self.population_size, self.
        ↪ iterations)

        solution = BGWO(X, y, self.population_size, self.
        ↪ iterations, self.test_size, self.random_state)

        sol = solution.bestIndividual.astype(int)
        is_empty_solution = np.sum(sol) == 0

        # select all features in case there was no solution
        selected_features_indexes = np.where(sol == (0 if
        ↪ is_empty_solution else 1))[0]

        self.support_ = np.array(sol, dtype=bool)
        self.n_features_ = np.alen(selected_features_indexes)
        self.selected_features_indexes_ = selected_features_indexes
        self.convergence_ = solution.convergence
        self.features_curve_ = solution.features
        return self

    def _get_support_mask(self):
        return self.support_

```

```

[ ]: from sklearn.compose import ColumnTransformer
    from sklearn.feature_selection import f_classif, SelectPercentile
    from sklearn.metrics import recall_score, precision_score, f1_score
    from sklearn.metrics import confusion_matrix

```

```
from sklearn.multiclass import OneVsRestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
```

These are the random numbers used to perform 30 experiments on NSL-KDD dataset

```
[ ]: split_randoms = [91, 568, 130, 416, 948, 314, 41, 294, 157, 777, ↵
↵ 298, 127, 450, 966, 597, 236, 351, 109, 510, 660,
                        490, 554, 814, 287, 971, 981, 333, 30, 555, 735]
```

These are the five classifiers used in the experiments

```
[ ]: classifiers = [(DecisionTreeClassifier, DecisionTreeClassifier()),
                    (GaussianNB, GaussianNB()),
                    (MLPClassifier, MLPClassifier(verbose=True)),
                    (KNeighborsClassifier, ↵
↵ KNeighborsClassifier(n_jobs=-1)),
                    (SVC, SVC(verbose=True))]
```

These are the main parameters that control the proposed approach

```
[ ]: # Algorithm parameters
multi_class = True
use_selector = True
use_gwo = True
percentile = 50
population_size = 20
max_iterations = 10
```

```

classifier_inst = DecisionTreeClassifier()
if multi_class:
    # enable parallelism
    classifier_instance = OneVsRestClassifier(classifier_inst,
↪n_jobs=-1)

```

First step in the proposed approach: split the data into 70% training and 30% testing

```

[ ]: # Read inputs
X_train, X_test, y_train, y_test =
↪read_data(multi_class=multi_class, rand=split_randoms[0])

```

Second step in the proposed approach: Preprocess training and testing datasets

```

[ ]: # Prepare nominal features transformation and numerical features
↪scaling
boolean_features_set = {'land', 'logged_in', 'is_host_login',
↪'is_guest_login'}
boolean_features = list(boolean_features_set.
↪intersection(set(X_train.columns.to_list()))))
numerical_ix = X_train.select_dtypes(include=['int64', 'float64']).
↪columns.drop(boolean_features)
categorical_ix = X_train.select_dtypes(include=['object']).columns
clmn_transformer = ColumnTransformer(transformers=[('num',
↪MinMaxScaler(), numerical_ix), # numerical features scaling
                                                    ('cat',
↪PdfTransformer(), categorical_ix)], # nominal features
↪transformation
                                                    remainder='passthrough',
↪n_jobs=-1) # keep the reset as is

```



```
# Prepare preprocessing pipeline
pipeline_steps = [('clmn_transformer', clmn_transformer)]
if use_selector:
    selector_classifier = SelectPercentile(f_classif,
    ↪percentile=percentile)
    pipeline_steps.append(('selector_classifier',
    ↪selector_classifier))
if use_gwo:
    bgwo_classifier = BgwoClassifier(population_size,
    ↪max_iterations, 0.10, 22)
    pipeline_steps.append(('bgwo_classifier', bgwo_classifier))
preprocessing_pipeline = Pipeline(pipeline_steps, verbose=True)

# Train preprocessing pipeline, and preprocess training data
X_train = preprocessing_pipeline.fit_transform(X_train, y_train)
# Preprocess testing data
X_test = preprocessing_pipeline.transform(X_test)
```

Third step in the proposed approach: perform classification training and testing

```
[ ]: # Train the classifier
classifier_inst = classifier_inst.fit(X_train, y_train)

# Test the classifier
y_pred_curr = classifier_inst.predict(X_test)
```

Print all the metrics

```
[ ]: # Print the metrics
if multi_class:
    conf_matrix = confusion_matrix(y_test, y_pred_curr)
    FP = conf_matrix.sum(axis=0) - np.diag(conf_matrix)
```

```

FN = conf_matrix.sum(axis=1) - np.diag(conf_matrix)
TP = np.diag(conf_matrix)
TN = conf_matrix.sum() - (FP + FN + TP)
fp = FP.astype(float)
fn = FN.astype(float)
tp = TP.astype(float)
tn = TN.astype(float)

# DoS -> 1, Probe -> 2, R2L -> 3, U2R -> 4
accuracy = accuracy_score(y_test, y_pred_curr)
recall_score_arr = recall_score(y_test, y_pred_curr,
↪average=None)
false_alarm_rate_arr = fp / (fp + tn)
precision_score_arr = precision_score(y_test, y_pred_curr,
↪average=None)
f1_score_arr = f1_score(y_test, y_pred_curr, average=None)

print('Accuracy: {acc}'.format(acc=accuracy))

normal_detection_rate = recall_score_arr[0]
normal_false_alarm_rate = false_alarm_rate_arr[0]
normal_precision = precision_score_arr[0]
normal_f1 = f1_score_arr[0]

print('Normal DR: {recall}, Normal FAR: {far}, Normal
↪Precision: {precision}, Normal f1: {f1}'
      .format(recall=normal_detection_rate,
              precision=normal_precision,
              f1=normal_f1,
              far=normal_false_alarm_rate))

```

```
dos_detection_rate = recall_score_arr[1]
dos_false_alarm_rate = false_alarm_rate_arr[1]
dos_precision = precision_score_arr[1]
dos_f1 = f1_score_arr[1]

print('DoS DR: {recall}, DoS FAR: {far}, DoS Precision:␣
↪{precision}, DoS f1: {f1}'
      .format(recall=dos_detection_rate,
              precision=dos_precision,
              f1=dos_f1,
              far=dos_false_alarm_rate))

probe_detection_rate = recall_score_arr[2]
probe_false_alarm_rate = false_alarm_rate_arr[2]
probe_precision = precision_score_arr[2]
probe_f1 = f1_score_arr[2]

print('Probe DR: {recall}, Probe FAR: {far}, Probe Precision:␣
↪{precision}, Probe f1: {f1}'
      .format(recall=probe_detection_rate,
              precision=probe_precision,
              f1=probe_f1,
              far=probe_false_alarm_rate))

r2l_detection_rate = recall_score_arr[3]
r2l_false_alarm_rate = false_alarm_rate_arr[3]
r2l_precision = precision_score_arr[3]
r2l_f1 = f1_score_arr[3]

print('R2L DR: {recall}, R2L FAR: {far}, R2L Precision:␣
↪{precision}, R2L f1: {f1}'
```

```

        .format(recall=r2l_detection_rate,
                precision=r2l_precision,
                f1=r2l_f1,
                far=r2l_false_alarm_rate))

    u2r_detection_rate = recall_score_arr[4]
    u2r_false_alarm_rate = false_alarm_rate_arr[4]
    u2r_precision = precision_score_arr[4]
    u2r_f1 = f1_score_arr[4]

    print('U2R DR: {recall}, U2R FAR: {far}, U2R Precision:␣
↪{precision}, U2R f1: {f1}'
          .format(recall=u2r_detection_rate,
                  precision=u2r_precision,
                  f1=u2r_f1,
                  far=u2r_false_alarm_rate))
else:
    conf_matrix = confusion_matrix(y_test, y_pred_curr)
    tn = conf_matrix[0, 0]
    fn = conf_matrix[1, 0]
    tp = conf_matrix[1, 1]
    fp = conf_matrix[0, 1]
    accuracy = accuracy_score(y_test, y_pred_curr)
    detection_rate = recall_score(y_test, y_pred_curr)
    false_alarm_rate = fp / (fp + tn)
    precision = precision_score(y_test, y_pred_curr)
    f1 = f1_score(y_test, y_pred_curr)
    print('Accuracy: {acc}, DR: {recall}, FAR: {far}, Precision:␣
↪{precision}, f1: {f1}'
          .format(acc=accuracy,
                  recall=detection_rate,

```

```
precision=precision,  
f1=f1,  
far=false_alarm_rate))
```