Anomaly Detection Using Multi Agent Systems

Sama Khosravifar

A Thesis

in

The Department

of

Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Master of Applied Science at Concordia University

Montréal, Québec, Canada

Fall 2018

Sama Khosravifar

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By: Sama Khosravifar

Entitled: **Anomaly Detection Using Multi Agent Systems**

And submitted in partial fulfillment of the requirements for the degree of

Master of Electrical and Computer Engineering

Complies with the regulations of the University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

________Dr. Anjali Awasthi ____________________________________ Chair

_______ Dr. Anjali Awasthi_____________________________________ Examiner

________Dr. Hassan Rivaz______________________________________ Examiner

________Dr. Wahab Hamou-Lhadj________________________________ Supervisor

Approved by:

______________Dr. W. Lynch__________________________________
Chair of Department or Graduate Program Director

Abstract

Anomaly Detection Using Multi Agent Systems

Sama Khosravifar

Daily access to Internet, increase in number of users, and newly discovered violations of policies, have become much more frequent over the last few decades as technology advances. Learning how to recognize these new violations as well as facing these new violations are two parallel concepts. There exist approaches that detect these violations often called intrusions or anomalies. A large body of knowledge focuses on developing new algorithms for anomaly detection, determining accurate thresholds for decision making upon detection, and combining different sources of data for increased performance. In this thesis, we propose a multi-agent anomaly detection system, in which agents collaborate with each other to detect anomalies in an effective way. We use multiple agents to set a cost on communication between them, and to make the final decision based on the combined results of all agents. Unlike other approaches, since our proposed approach is flexible in terms of the number of agents, so it will not fail while using fewer agents, or some agents fail to perform.

The key elements in our approach are in using system call based datasets, deciding on the number of agents, and their methodologies, as well as the cost for communication between the agents. The final result of the system might ignore agents if they are not providing feedback that will result in higher accuracy of anomaly detection. We analyze the results by plotting a Receiver Operating Characteristic (ROC) curve and focusing on the Area Under the Curve (AUC) using different thresholds. We make the final decision based on the most suitable threshold for agents.

Table of Contents

List of Figures

List of Tables

List of Equations

Chapter 1 - Introduction

1.1. Context and Motivation

Over the past few decades, daily access to Internet has become not just easy, but ubiquitous as a result of the vast improvements made in computer systems and network technology. This has contributed to a constant growth in the number of users. As a result, we also witness a vast array of security threats in computer systems [1] [2].

These threats manifest themselves as system intrusions, bypassing the security of a computer system and network. This explains the emergence of Intrusion Detection Systems (IDSs) [3], which aim to detect and mitigate intrusions.

1.1.1. Intrusion Detection Systems (IDSs)

IDSs are software applications for detecting system behaviors that deviate from a known normal behavior. These tools act in different ways using various techniques [3].

1.1.2. Anomalies

An *anomaly* or *intrusion* is an event, or a pattern of events, that deviates from the expected normal behavior of the system [5]. To detect anomalies, there are different approaches that vary depending on the type of dataset, the type of anomalies, as well as the overall detection algorithm.

1.1.3. Detection Methodologies

1.1.3.1. Signature-based Anomaly Detection

Signature-based detection is also known as *knowledge-based detection* or *misuse detection*, operates on known attacks, depicted using signatures [6-8]. A tool that supports this technique compares system execution with attack signatures, and raises a flag if a match exists. Signature-based anomaly detection techniques are limited to known attacks, meaning that they cannot detect zero-day exploits.

1.1.3.2. Anomaly-based Anomaly Detection

Anomaly-based detection is also known as *behavior-based detection*. Anomaly-based approaches start by building a model based on normal patterns of dataset [8]. The dataset at this step is collected within a period of time in an attack-free environment [9]. Once the model is ready, the anomaly detection process starts by looking for significant deviations from the normal patterns [10-12]. This deviation does not necessarily relate to an attack; it can be a normal event which does not happen often [13-14].

Some of the machine learning techniques, for instance, Artificial Neural Networks or Support Vector Machines can be used for anomaly detection. In order to build a model out of normal events, the requirement of the algorithm is to deal with fixed-length patterns. Hence, the dataset is usually divided into smaller fixed-length events [15-19].

1.2. Objectives

The objective of this thesis is to use Multi-Agent Systems (MAS) [20-22] to enhance the accuracy of detecting anomalies in host computers. Each agent implements a unique detector, usually based on a machine learning algorithm. Agents collaborate with each other dynamically for improved accuracy.

1.3. Focus on System-Call Traces

In this thesis, we focus on using system call traces as the main source of data. System calls provide an interface between a user and the kernel of the operating system [22-23]. Every system call has a unique number that is defined by the kernel. Users' requests are made from higher level applications to the kernel via a set of system calls and arguments. Some examples of system calls used for file management are *open()*, *read()*, *write()*, and *close()*.

A combination of these system calls collected over a period of time is called a *trace* or a *stream* [22] [24]. These traces have different lengths that mainly depend on the complexity and execution time of the traced process. System call traces have been using extensively in anomaly detection research [9] [15] [26-27]. This is because system calls are usually difficult to be altered with and, hence can be considered a reliable source of data. Although we focus on system call traces in this thesis, we believe that the multi-agent architecture presented in this thesis is readily applicable to other types of data.

1.4. Thesis Contribution

The objective of this thesis is to improve the accuracy of anomaly detection using a multi-agent architecture in which agents support different detectors. The communication between agents is designed in a way that improves the final decision.

1.5. Thesis Outline

The rest of the thesis is structured as follows:

Chapter 2 - Background

This chapter is a walk through related research studies that shaped the ideas behind this thesis. We focus on anomaly detection techniques that are based on multi-agents systems.

Chapter 3 - Methodology

In this chapter, we present our methodology using a multi-agent architecture for detecting anomalies.

Chapter 4 – Evaluation

This chapter is dedicated to evaluating our approach. To this end, we experiment with two large datasets.

Chapter 5 - Conclusion and Future Work

We conclude the thesis and provide future directions in this chapter.

Chapter 2 –Related Work

2.1. Agent

An agent is an intelligent entity that is capable of completing an assigned task through its own programming, i.e., relying on itself [25]. Agent-based systems are usually composed of various types of agents. According to *Nwana's* categorization [26], represented in Figure 1, there are four types of agents that exist, explained in the following.

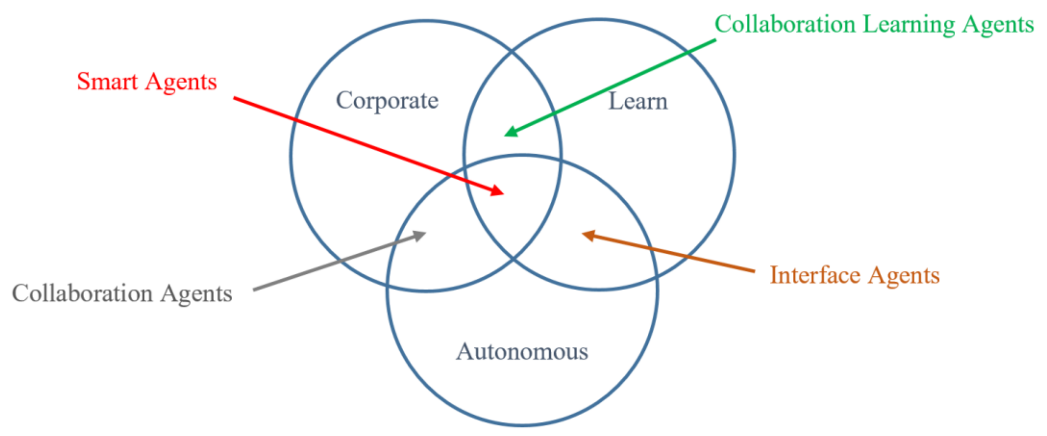


Figure 1. Nwana's categorization of agents [26]

- Collaboration Agents: These agents are more focused on cooperating and collaborating rather than learning from other agents.
- Interface Agents: These agents focus more on the learning and doing the task on their own.
- Collaborative Learning Agents: These agents learn from each other and collaborate, in order to cover their task.

- Smart Agents: These agents are called smart because they cover all three aforementioned factors.

2.2. Multi-Agent System (MAS)

A Multi-Agent System (MAS*)* [20-22] is a computerized system composed of multiple interacting agents within an environment. They are often used in complex problems that are difficult or perhaps even impossible for an individual agent to solve [19]. One benefit of using MAS is the flexibility of adding and removing autonomous agents as well as assigning new tasks depending on the context [19] [27].

MASs are used in different researches with the purpose of implementing the collaboration between entities. Some examples of the usage of MASs are in enhancing the security of automation networks [28] or making better decisions in an environment of more than one intelligent entity [29].

In a MAS context, agents independently function and communicate with each other within the system. We use MAS in this thesis for the purpose of hosting rational agents in order to detect anomalies using multiple approaches.

The literature in MAS falls into two main directions:

- The way agents are deployed to complete tasks on their own.
- The communication among agents to collaborate on specific tasks.

In addition, in [30], Becker et al. describe other concepts related to MAS, which are relevant to this thesis:

- Markov Decision Process. MDPs are mathematical frameworks used for the purpose of decision making processes, and widely used in the areas of robotics and automatic control. One of the main characteristics of MDPs refer to the fact that the status of an entity at time *t* only depends on its status of it at time *t-1*. Based on the factors necessary in the process of decision making of a system, the MDPs fall into different categories.
- Centralized MDP: In this type of MDP, the decisions made by entities depend on the status of the whole system and not just the status of the entity itself. Within these MDPs, all entities play a role in the decisions made for the whole system. If the result of one entity does not contribute in improving the overall system’s goal or performance, it would prevent the system from obtaining the best results, or causing a delays in achieving the system’s goal.
- Decentralized Markov Decision Process (DEC-MDP): These MDPs are more suitable for planning the decision making process within a dynamic system composed of multiple entities. Decisions made by DEC-MDPs are based on the partial knowledge of the whole system, and mainly depending on the status of entity or agent at each event.

2.2.1. Transition-Independent DEC-MDP

In [30], Becker et al. proposed a model of two agents that are involved in the decision making process. These agents act independently to provide feedback to each other based on a reward structure, which depends on the history of the agents. In other words, any agent involved in an event will have that event in its history and these histories are one of the factors involved in the decision making process of the whole MAS.

The authors use one algorithm to maximize the global value of the joint policies. Since there are two agents involved, there are two MDPs, as well as some boundaries for the policies of MDPs. The main target is to find the possible joint policies based on the history of agents and choose the one which results in maximum joint policies.

In [29], Becker et al. go deeper into the structure of the MDPs. The authors use two NASA rovers to monitor their surroundings as they move to different sites to take pictures, and conduct experiments. The collected information is sent back to NASA ground control. The authors use an augmented MDP for each agent. Agents can affect the policies of the other agents' MDPs. The aim of using an augmented MDP is to obtain the feedback of other agents involved in the decision making process. The results show that this design yields better policies for maximizing the information required by the ground control.

2.2.2. Communication Decisions in Multi-Agent Systems

In [31], Xuan et al. propose an approach in which the focus is on the communication aspect of agents within a MAS, which has an associated cost at each communication step. This cost of communication is used by agents to decide on the type of communication and it will affect the MDP of the agent. In this paper, the authors experimented with two robots, represented by two agents, which were instructed to meet within a specified time range in a 4*4 grid world. The robots are rewarded or penalised based on their history. Decisions at each step are based on the cost of the communication, the reward of the move, and passing the threshold assigned to the history of the agent.

There is a threshold considered for the cost of communication that decides for the rest of the approach:

- If the cost is lower than the threshold, they will focus on the history of the most recent communication. At the end of each communication, the distance between the agents and the distance from the target should be checked for further decisions.
- If the cost reaches the threshold or exceeds it, agents will ignore the communication and act on their own, meaning that they ignore each other.

In [32], Allen et al. used two agents to move two small boxes alone to a target area and one large box together as a team. The algorithm used in this paper focuses on assigning initial values towards actions that would help agents achieve their joint goal. These initial values make one agent ask for feedback from the other agent and then try to choose the actions that can be done with the closest agent in the neighborhood.

In [33], Minker et al. designed two agents to present as rovers on Mars to collect information for the ground control. Because of resource limitations (computer power, memory, et.), the agents are designed to operate autonomously until communication with the control center becomes feasible.

The decision making process in this paper is based on the DEC-MDP approach. The authors consider a transition and reward function for agents in order to increase the chance of making better decisions.

2.2.3. Agent Community Scalability

Although these studies use multiple agents, depending on the proposed methodology, the flexibility in varying the number of agents may be difficult to achieve.

Because the history of agents plays an important role in finding the maximum joint policies, increasing the number of agents would exponentially increase the complexity of considering the impact of each agent's history in reaching the main goal.

In [29], the proposed methodology involves using an augmented MDP, that at each step, all agents have an influence on the decisions made per agent. Hence, increasing the number of agents adds complexity and takes more time to reach to the final target.

In [31], according to the proposed methodology, the decisions made by one agent highly depend on the distance from the other agents and from the final target. Involving more agents adds complexity in the decision making processes. Also, agents might face situations with more than one direction to go to, as a result of having two agents in the same distance, which may hinder effectiveness.

In [32], the fact that one of the assigned tasks requires both agents to function, makes the model not flexible enough as to the choice of selecting the number of agents. The increase in the number of agents can have positive impact because the model can decide which agents can act better as part of a team work, but if the number of agents decreases to two, there is no option to choose from. The model fails if only one agent is used.

In [33], according to the limitations for the rovers, the proposed approach is not flexible enough in terms of number of agents. Hence, adding more agents to the model might cause failure in the whole process of collecting data.

2.2.4. History/Memory Scalability

In [32-36], although the proposed models reach to their desired target, making some changes to the model may cause the model to run out of memory. For example, increasing the number of agents would mean an increase in the number of MDP histories. Processing large histories require computational resources.

In [33], although the proposed architecture suffers from memory limitations, but since both rovers receive orders from the ground control, a potential solution would be to stop one of the rovers, or to delete part of the history of rovers.

2.2.5. Cost of Communication

In [30], the proposed model assigns a weight for communication between the agents. This cost or reward guides agents towards making better decisions in terms of choosing when to communicate or give feedback to the rest of the agents.

In [29], instead of having a communication system between the agents, the proposed model functions based on an augmented MDP, in which at every step, agents have an impact on the MDP of the other agents.

In [31], the methodology of the system is on the basis of assigning a cost for communication between the agents, as well as checking this cost with a threshold. In other words, the model proposed in this paper, limits agent communication if the cost of communication is more than a predefined threshold.

In [32], according to the structure of the system, there is a cost considered for communication especially for the task assigned to both agents as a team work.

In [33], the communication between the rovers is controlled from the ground control. Since the main goal of these rovers is to collect as much information as they can, the choice of communication becomes necessary.

Chapter 3 – The ADUMAS Approach

3.1. Background

As discussed in the first chapter, an agent is an individual entity, which receives data from the surrounding environment to complete an assigned task. According to *Nwana's* categorisation of agents [26], smart agents go through a training phase by obtaining information from their environment to take action or decide on whether they want to cooperate with other agents.

Under different circumstances, among which are the environment and the type of the dataset being used for said training phase, there might be some uncertainty about the final result of the agent. At this point, trust becomes necessary. There should be some factors taken into account, putting forth the level of trust regarding the output of the agent. These factors depend on the dataset and the used methodology.

In a MAS, agents are trained individually. They can decide whether they want to share their individual point of view with the rest of the agents, as well as how much they trust each other's results. Looking at a bigger picture, and based on the status of agent at the time, the decision might fall into these categories:

- Considering other agents' feedback and using their influence on individual result of the agent.
- Ignoring the feedback and continuing to complete the task solely.
- Depending on the task assigned to the agent and the feedback received, the agent might go through a combination of the aforementioned options at different steps.

The main purpose of using a MAS is to reach a more reliable result, or in other words, a higher level of trust in the final outcome of the system.

Our proposed approach is based on using a MAS to detect intrusions. We implement a system composed of three agents, each containing three modules, and one module outside of the agents that communicate with all three agents. We discuss our proposed architecture, called ADUMAS (*Anomaly Detection Using Multi Agent Systems*), in the next section.

3.2. Structure of ADUMAS

ADUMAS consists of three anomaly detection agents, which implement three known anomaly detection techniques, mainly Sequence Time Delay Embedding (STIDE) [9], K-Nearest Neighbor (KNN) [35] and Graph Edit Distance (GED) [36] respectively.

Tasks assigned to each agent are described as follows:

- Training phase: Each agent uses the entry dataset and, based on the supported technique, it learns the normal pattern of behavior.
- Testing phase: An agent starts comparing a new pattern with known patterns with the objective of detecting anomalies.
- Updating Phase: At the time all agents of the same environment reach their individual results, they might need to give feedback to each other. This feedback may require some agent to update their knowledge on the normal patterns.

Although an agent functions on its own, regardless of the methodology it is assigned with, its combination of three aforementioned phases means that each will be considered as a module. Hence, there will be three modules per agent within the environment.

According to the structure of STIDE, KNN, and GED, the entry dataset should be in the format of separate sequences of system calls. Since there are different types of datasets, depending on the one we use, the datasets should be in a format that fits into all agents. So we are considering one separate module outside of the agents, making the required changes in the entry dataset.

The general representation of ADUMAS can be found in the Figure 2.

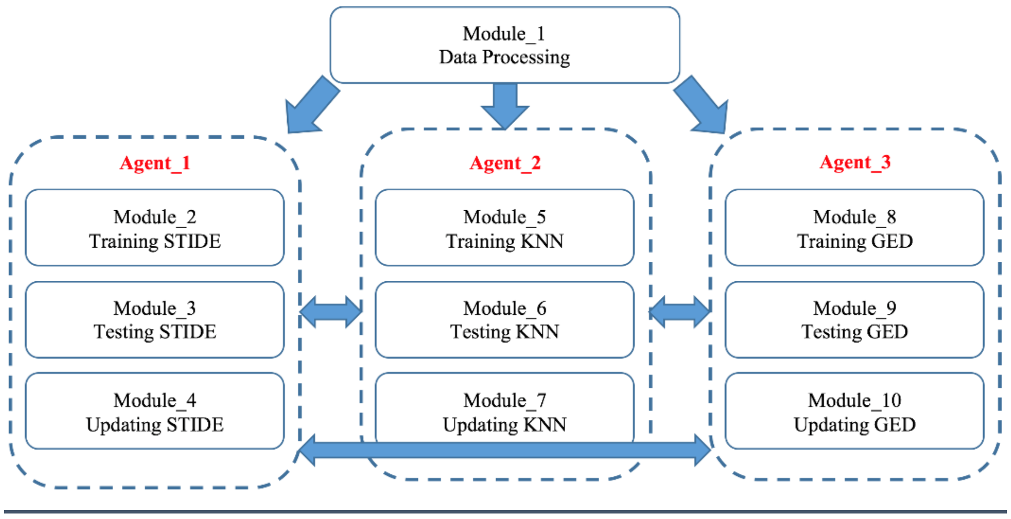


Figure 2. ADUMAS structure

Once the algorithms involved are chosen and there is a dataset to work with, the *data processing module* receives the dataset, prepares the data in terms of traces of system calls, and passes them to all agents. Each agent uses the training traces for the training phase, and uses the attack traces in the testing phase. In other words, the results of individual agents are ready as the output of the *testing module*. After having all sole results, the *updating modules* start analyzing the results, giving feedback to the rest of the agents, and based on the received feedback from other agents considers some process to update the results. This process of giving feedback and updating the results will continue on through a loop until the agents agree on the final results.

We will further clarify this procedure after knowing more detail about the modules, agents, the methodologies used in ADUMAS, and the connection between the agents.

We believe that ADUMAS can be more effective than existing models that focus on either anomaly detection using a single methodology, or using a vote-based approach. It improves the accuracy of detection, which is the objective of this thesis. We are not just using agents within ADUMAS for voting, it is more of a dynamic decision making system with respect to certain criteria. We are using artificial intelligence techniques to let agents vote for label of traces with certain confidence level, which is the result of the technique they use and the threshold they set. Using ADUMAS, the final decision as to whether an anomaly is detected or not considers the following scenarios:

- All agents agree on the label, but they have different confidence levels bout the trust level they are putting in their own decision. Thus, the final decision of ADUMAS will be concluded based on joint results of the agents.

- One agent disagrees with the rest of the agents regarding the labelling of normal versus abnormal. In this case, ADUMAS will ignore the results with a low trust level and will make the final decision based on the results achieved with a higher confidence level. ADUMAS will not take the side of two agents just because they voted for a specific label, but with low level of confidence. The fundamental point in this thesis is to reach a decision with a high confidence level.

3.3. Datasets

3.3.1. UNM Datasets

The UNM, which stands for University of New Mexico, is among the most commonly used system call based datasets for intrusion detection [37]. In order to detect anomalies, we need a model trained by normal traces collected during an attack-free request to kernel services. Following this, we will be able to detect the intrusions that are collected during the requests made to kernel services under different types of attacks, such as decode attack, buffer overflows, and more.

- Training traces: There are 13,726 normal traces with different lengths.
- Attack traces: There is an attack traces with 205,929 system calls.

The normal traces are collected within an attack-free operation in a secured environment, while the attack traces were collected in a non-attack-free environment.

3.3.2. ADFA Linux Dataset (ADFA-LD)

The ADFA-LD datasets, which stands for *Australian Defence Force Academy Linux Datasets*, is publicly available on the website of University of New South Wales (NSW) [38]. This dataset is composed of the following parts:

- Training traces: There are 833 normal traces with different lengths and in total there are 308,077 system calls.
- Validation traces: There are 4373 normal traces with different lengths and in total there are 2,122,085 system calls.

- Attack traces: There are 746 attack traces with different lengths and in total there are 317,388 system calls.

The normal traces are collected within an attack-free operation in a secured environment, while the attack traces were collected in a non-attack-free environment.

3.4. Anomaly Detection Algorithms

In this thesis, we chose to use three well-known anomaly detection algorithms: STIDE, KNN, and GED.

3.4.1. STIDE

In [9], Khreich etal. refer to the fact that while sending requests to kernel services, the order of its response in the form of a sequence of system calls can play an important role in describing the normal behavior of the made request. One of the approaches in analyzing datasets containing sequential system calls is STIDE, standing for *Sequence Time-Delay Embedding*, and based on building *N-gram* models.

N-gram models [9][39] are vectors of *N* system calls extracted from a trace of system calls. Regardless of the size of trace, this model deals with breaking the trace into fixed length sequences of size *N* system calls.

This model breaks down the normal traces into *N-gram* sequences and collects the unique sequences. Later, for the purpose of detecting anomalies, it compares the attack *N-grams* with the built model. Regardless of the value of *N*, the important part is to deal with fixed length sequences. This means that by using greater values of N, the model can store larger responses from the kernel services.

By using this methodology, we build a model out of normal behaviors by sliding a window of length *N* on normal traces. Then extracts the sequences of length *N* using the same procedure from unknown behavior, and compares them with the normal model. New traces being closer to the normal behaviors within the model, refers to being more prone to be normal.

Among the different values used for N as the size of sequences, studies show that *6-grams* have higher detection accuracy [9]. Hence, we use the 6-gram models in the first agent. This falls into two main steps:

- In the first step, we extract sequences of length six from normal traces in order to train a model. This will be done by means of sliding a window of length six on traces until we reach the end of the main trace. Performing this process on all of the normal traces, we will end up obtaining a model that can be used for detecting the anomalous traces.
- In the second step, if the dataset contains validation traces we test the built model using these traces and in case a dataset doesn't have this category, we randomly choose 70% of training traces for building the model and use the rest of it as a validation set. The main intention is to test the model and update it according to the validation set before using it for detecting the anomalies.
- In the third step, we do the same extraction of sequences of length six, but this time on the attack traces. In this step, each one of the extracted sequences must be compared with the trained model. Whenever there is a match with the model, the sequence will be labeled as normal, otherwise it will be known as anomalous sequence. It is worth mentioning this step doesn't finalize the detection process. According to the structure of the ADUMAS, labels of this step will be used in order to finalize the detection process. We assign two variables, *normalCount* and *anomalyCount*, to each attack trace so that we

can keep track of number of normal and abnormal extracted sequences of that very trace. If the sequence already exists in the model, we incrementally increase the *normalCount* by one, otherwise we incrementally increase the *anomalyCount* by one. At the end, for each attack trace, it will report the number of *normalCount* and *anomalyCount* regarding that very trace. The following example will clarify this part.

3.4.1.2. Example

In order to illustrate how STIDE works, we will use it to detect the normal and anomaly fixed length sequences of an attack trace given two normal traces, in the example represented in Figure 3 and Figure 4.

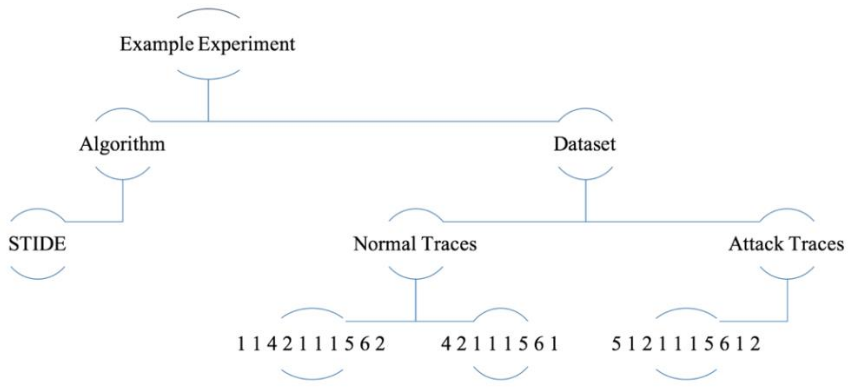


Figure 3. Example experiment

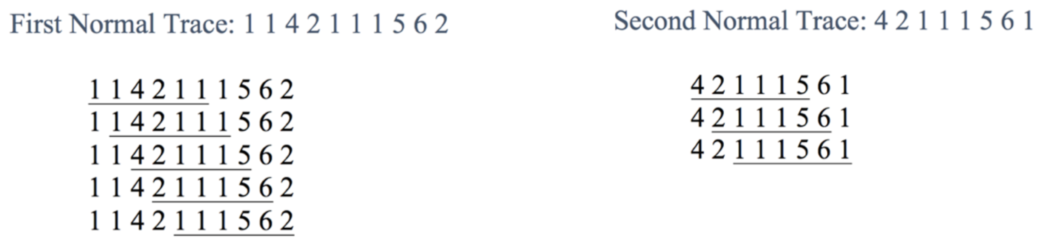


Figure 4. Break down of normal traces in STIDE

Afterwards, the sequences must be added to the model. Table 2 represents the status of each sequence, whether that every sequence can be added to model, or whether it should be ignored while it already exists in the model.

Table 1. Status of extracted sequences of normal traces in STIDE

Sequences	Status
1 1 4 2 1 1	Add to the model
1 4 2 1 1 1	Add to the model
4 2 1 1 1 5	Add to the model
2 1 1 1 5 6	Add to the model
1 1 1 5 6 2	Add to the model
4 2 1 1 1 5	Ignore the sequence
2 1 1 1 5 6	Ignore the sequence
1 1 1 5 6 1	Add to the model

After covering all of the normal traces and extracting the fixed-length sequences and labeling their status, the model will be ready to be used for detection.

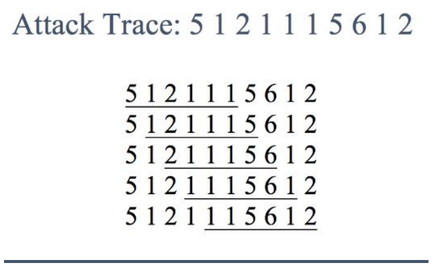


Figure 5. Break down of attack trace in STIDE

Next, the built model will be used in order to detect the sequences of the attack trace. According to Table 3, the sequences that fail to match with the model are the *anomalies.*

Table 2. Status of extracted sequences of attack traces in STIDE

Sequences	Status
5 1 2 1 1 1	Doesn't exist in model
1 2 1 1 1 5	Doesn't exist in model
2 1 1 1 5 6	Exists in the model
1 1 1 5 6 1	Exists in the model
1 1 5 6 1 2	Doesn't exist in model

The label that will be assigned to the whole trace depends on the number of normal and anomaly sequences of that trace. This is further discussed in the next subsection.

3.4.1.3. Output of STIDE

We present the output of the *testing module* in the form of the name of the attack trace, and the number of normal 6-grams and anomalous 6-grams within each trace.

In case there was only one approach and it was using the *N-gram* models, we process these results and detect the traces, but since there are two other agents working on the same dataset, we consider this result as an individual analysis of the first agent.

We find the individual result of the other agents and then we trigger the *updating module* to continue the analysis and finalize the anomaly detection process. The updating part is explained in Chapter 4.

3.4.2. KNN

The second agent is assigned to work with the *K-Nearest Neighbor* algorithm, also known as KNN. We use this algorithm for classification purposes. The main idea is to build one class of data points and then compare the rest of the data points with this class. Since we build only one class, this means that we set K to 1.

3.4.2.1. KNN Data Points

The second agent receives traces of system calls as the output of the *data processing module.* We consider each one of these traces as one data point, regardless of them being normal or attack.

Since, in the first methodology, in order to give a label and weight to the attack traces, we broke them into subsequences of length six and estimated the results, we do the same in the second methodology. After dividing each trace into traces of length six, for each subsequence we estimate the value of five features. The average of these five features will be assigned to the whole trace.

The idea behind this methodology is to give more attention to some patterns that can be found within the normal traces. For instance, the high frequency of seeing some system calls within the normal traces, the high frequency of facing a transition between two specific system calls, or any other patterns.

In the following subsections, we represent this process, in details. All of these steps must be covered for normal traces, which results in having values of five features assigned to each normal trace. The values from all normal traces form the range of models that we use as one class of this algorithm. We proceed in the same way with attack traces and compare the results of five features with the normal range of models.

3.4.2.2. First Step (Sequence Extraction)

In the first step, for each trace, we extract sequences of length six. This is done in the same way as in STIDE, i.e., by sliding a window of length six on the trace, extracting the sequences, shifting by one to the right on the trace, and extracting the next sequence. This goes on until the window reaches the end of the trace. It is worth mentioning the traces with small size (less than six system calls) are ignored, because they cannot even provide a single 6-gram sequence for this algorithm.

In order to clarify the process, we follow an example of three traces, given that the alphabet of the dataset is {1, 2, 3, 4}. In Figure 6, the breakdown of the traces is presented.

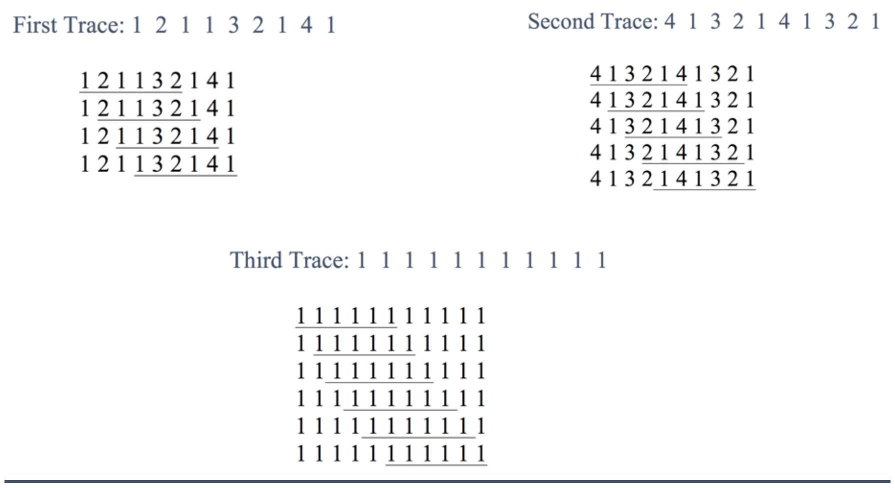

First Trace: 1 2 1 1 3 2 1 4 1
Second Trace: 4 1 3 2 1 4 1 3 2 1
Third Trace: 1 1 1 1 1 1 1 1 1 1 1

Figure 6. Breakdown of sample traces in KNN

3.4.2.3. Second Step (Frequency of Subsequences)

In the second step, the frequency of subsequences should be estimated. This frequency only considers the sequences of traces belonging to the same type; either they belong to normal traces or belong to attack traces.

Table 3. Frequency of subsequences of sample traces in KNN

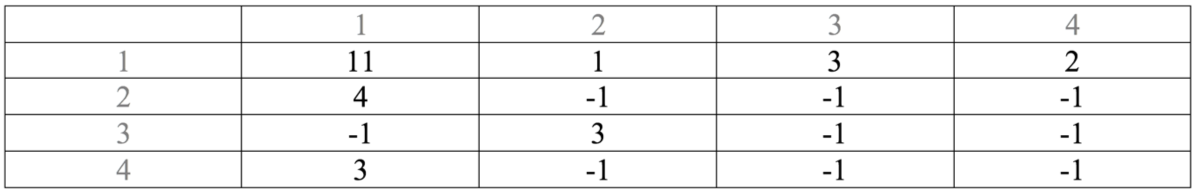
Subsequences	Frequency
1 2 1 1 3 2	1
2 1 1 3 2 1	1

1 1 3 2 1 4	1
1 3 2 1 4 1	2
4 1 3 2 1 4	1
3 2 1 4 1 3	1
2 1 4 1 3 2	1
1 4 1 3 2 1	1
1 1 1 1 1 1	6

3.4.2.4. Third Step (Transition Frequency of System Calls)

In the third step, we fill out a two dimensional matrix called the *transition matrix* [n+1]*[n+1], in which n represents the size of the alphabet. This table contains the frequency of transitions from each system call to the rest of the system calls. In case there is no transition from a specific system call to another one, the value of transition frequency assigned set to -1. Table 5 shows an example of this table when used with traces of the previous subsection.

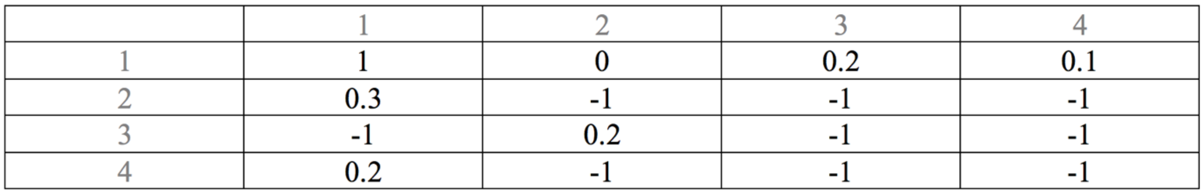
Table 4. Transition matrix sample in KNN



	1	2	3	4
1	11	1	3	2
2	4	-1	-1	-1
3	-1	3	-1	-1
4	3	-1	-1	-1

The first row and first column contain all of the system calls of the dataset, or in other words, the alphabet of the dataset. Once the table is completed, all of the values except the -1 must be normalized. The completed table is presented in Table 6.

Table 5. Normalized transitions of sample in KNN



	1	2	3	4
1	1	0	0.2	0.1
2	0.3	-1	-1	-1
3	-1	0.2	-1	-1
4	0.2	-1	-1	-1

3.4.2.5. Fourth Step (Frequency of System Calls)

In the fourth step, we sort the system calls on the basis of their frequency. Table 7 contains all of the system calls sorted based on their frequencies.

Table 6. Frequency of system calls of sample traces in KNN

System Calls	Frequency
1	20
2	4
3	3
4	3

Afterwards, the sorted system calls must be given indexes starting from one. In other words, system calls with the smallest index, represent the highest frequency in the dataset. Also the system calls with equal value of frequency must be given one index because there is no factor discriminating between them, hence, they must have the same priority.

Table 7. Assigning index to system calls of sample traces in KNN

System Calls	Frequency	Index
1	20	1
2	4	2
3	3	3

4	3	3

3.4.2.6. Fifth Step (Evaluation of Features)

In the fifth step, we make use of the previous steps in order to evaluate five features to assign to each sequence of length six. This way, we have a better understanding of data points by looking at them from diverse angles and considering all of the aspects in making a decision on their labels. In the following section, we estimate the features for the sample sequence {1 2 1 1 3 2}.

3.4.2.6.1. First Feature

The first feature is called *NormalizedX*. For each subsequence, named X, we need the following factors in order to estimate this parameter:

- Frequency of X
- Number of sequences of the same type (either normal or attack)

In other words, this parameter is the normalized version of a sequence's frequency, so it will be in the range of [0, 1]. According to the formula, if the frequency is close enough to the total number of sequences, it results in a value closer to 1, referring to the fact that the dataset is mostly composed of this very sequence.

$$NormalizedX = \frac{Frequency(X)}{Number\_of\_sequences\_of\_the\_same\_type}$$

Regarding the example, this parameter would be:

$$X = sequence\,\{1\ 2\ 1\ 1\ 3\ 2\}$$

$$NormalizedX = \frac{Frequency(X)}{Number\_of\_sequences\_of\_the\_same\_type} = 0.0667$$

3.4.2.6.2. Second Feature

The second feature is called *Average_Transition_Frequency*. Since each subsequence is composed of six system calls, it results in five transitions within that sequence. The *Average_Transition_Frequency* parameter is the average of five normalized transitions mentioned in the table of third step, while ignoring the -1.

The evaluation is clarified in the formula below. This parameter is also ranged in [0, 1] since we use the normalized values of transitions.

$$Average\_Transition\_Frequency = \frac{1}{5}\sum_{m=1}^{5} Normalized\_Transition_m$$

Regarding the example, this parameter will be:

$$X = sequence\ \{1\ 2\ 1\ 1\ 3\ 2\}$$

$$Normalized\_Transition = [0, 0.3, 1, 0.2, 0.2]$$

$$Average\_Transition\_Frequency = 0.34$$

3.4.2.6.3. Third Feature

The third feature is called *Minimum_Transition_Frequency*. Each sequence contains five transitions between its system calls and their values, and can be found from the *Normalized_Transition* table, while ignoring the -1.

3.4.2.6.4. Fourth Feature

The fourth feature is called *Average_Index_Frequency*. This parameter is the average of five indices assigned to the system calls of the sequence. Considering the general case where the size of alphabet is n and the indexes are integers from 1 to n:

- The minimum value of this parameter can be reached where the indexes of the sequence are all equal to 1. This means the sequence contains the system calls with the highest frequency.
- In terms of the maximum value of this parameter, the sequence must contain the system calls with the least amount of frequency, given the fact that there are no equal frequencies, so the indexes will be equal to n in which the *Average_Index_Frequency* results in value of n.

So, this feature is ranged in [1, n]. Regarding the example above, the value of this parameter will be:

$$X = sequence\ \{1\ 2\ 1\ 1\ 3\ 2\}$$

$$system\ calls = \{1,2,3\}$$

$$indexes = \{1,2,3\}$$

$$Average\_Index\_Frequency = \frac{1+2+1+1+3+2}{6} = 1.667$$

3.4.2.6.5. Fifth Feature

The fifth parameter is called *Sq_Rt_Of_Sum_Of_Squares* (Square Root of Sum of Squares). The same as fourth feature, it also deals with the index of the system calls, but we are putting more weight onto indexes by using the square value of them.

$$Sq\_Rt\_Of\_Sum\_Of\_Squares = \sqrt{\sum_{i=1}^{6} I_i^2}$$

In order to find the range of this parameter, we consider two scenarios.

- The minimum value of this feature can be reached where the indexes of the sequence are all equal to 1. This means the sequence contains the system calls with the highest frequency. Hence, the minim value will be $\sqrt{6}$.
- Regarding the maximum value, the sequence must contain the system calls with the least amount of frequency, given the fact that there are no equal frequencies, so the index will be equal to n in which the *Average_Index_Frequency* results in value of $n\sqrt{6}$.

So, this feature is ranged in [$\sqrt{6}$,n$\sqrt{6}$]. Regarding the example above, the value of this parameter will be:

$$X = sequence \; \{1\ 2\ 1\ 1\ 3\ 2\}$$

$$system\ calls = \{1,2,3\}$$

$$indexes = \{1,2,3\}$$

$$Sq\_Rt\_Of\_Sum\_Of\_Squares = \sqrt{\sum_{i=1}^{6} I_i^2} = \sqrt{1^2 + 2^2 + 1^2 + 1^2 + 3^2 + 2^2} = 4.472$$

3.4.2.7. Sixth Step (Feature Values of Sequences)

Once the five features of sequences of length six are estimated, the per feature value of each trace should be assigned to it by getting an average of that feature from the subsequences belonging to that trace. This results in having five values for each trace.

3.4.2.8. Seventh Step (KNN Output)

After covering all the above steps for normal and attack traces separately, we have a normal range of features (model) and we should figure out how far away are the attack traces from this class of data points. This is mainly the *testing module* of the second agent.

In this phase for each trace, if at least three out of five features are within the normal range, that trace will be considered normal.

3.4.3. GED Algorithm

The third agent is assigned to work with GED, which stands for *Graph Edit Distance*. This methodology is mainly used to find the similarity between two graphs. The main idea behind this algorithm is to turn one graph into another graph by using minimum operations. A set of these operations are insertion, deletion, and substitution of both nodes and edges.

3.4.3.1. Graphical Example of GED

In the GED algorithm, we deal with two graphs called the source graph and target graph. The main idea is to turn the source graph into the target graph by way of as few operations as can be managed. Consider the following example presented in Figure 7, in which the source graph is composed of four nodes and four edges, and the target graphs is composed of four nodes and three different edges.

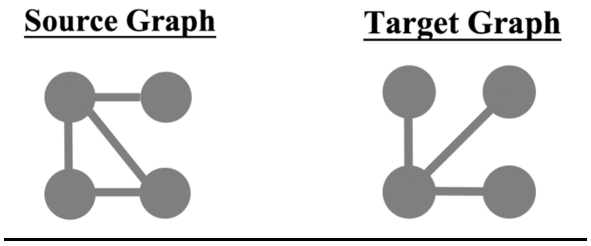


Figure 7. Source and target graphs

Figure 8 represents the process of turning the source graph into the target graph, which demands three operations in total in order to complete the process of transition.

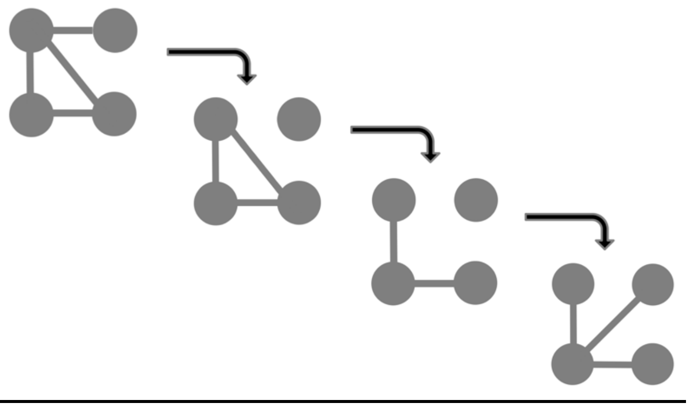


Figure 8. Transition of source graph into a target graph

3.4.3.2. Converting Traces into Graphs

In ADUMAS, each agent deals with many traces of system calls. Since GED works with graphs, we must turn the traces into graphs, which is the initial phase of the algorithm. Each trace is a string of system calls that can be considered a graph, in which the nodes represent the system calls, and the edges refer to the transition between one system call to the other.

Figure 9 shows an example of converting a trace of system calls into a graph.

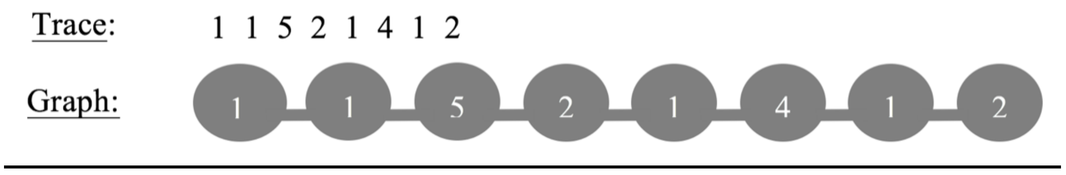


Figure 9. Converting trace into a graph

It the structure of ADUMAS, the only operation we consider is deletion. For the datasets with a small alphabet size, we can consider the edit or inset operators as well, but since in this thesis we deal with a large alphabet, using the edit or insertion operators can complicate the calculations, hence we focus on using the deletion to match the source and target graphs. In other words, the distance between two graphs is calculated by the minimum number of deletions required to turn the source graph into the target graph.

3.4.3.3. Table of Distance

GED evaluates the distance between source graph of size n, and target graph of size m, by means of completing a table of size (n+2) *(m+2), called the *table of distance*. The contents of the table are as follows:

- The top left cell is empty.
- The cell on the right side of the empty cell is equal to э, as well as the cell at the bottom of the empty cell.
- The first row and the first column starting from third cell contain the nodes of source graph and target graph, respectively.

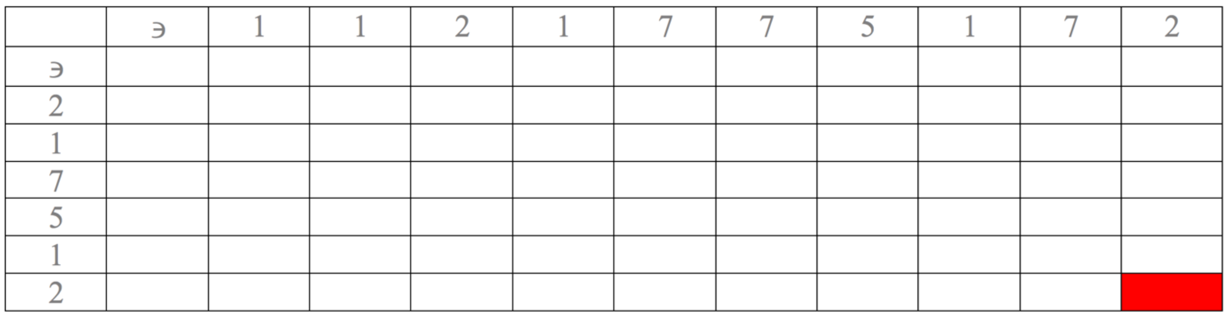
Table 8. Table of Distance

	э	Nodes of source graph		
э				
Nodes of target graph				

3.4.3.4. Completing the GED Table

In order to clarify the process of completing the table of distance, we use an example. In Table 10, the sample source graph and target graph are inserted into the table.

Table 9. Insertion of nodes of source graph and target graph



	э	1	1	2	1	7	7	5	1	7	2
э											
2											
1											
7											
5											
1											
2											🟥

We complete the table of distance according to rules of GED. The red cell in the table holds the final result, which is the minimum difference between the source graph and target graph. There are three different situations that might be encountered:

- If the graphs are the same, the red cell will contain zero, since they are the same graph.
- If the graphs are totally different, meaning they don't have any nodes in common, the value assigned to the red cell must be equal to the size of bigger graph. In other words, all of the nodes were removed and still there was no match with the other graph.
- If the graphs share at least one node, we follow the rules of GED to find the distance between the two graphs.

Rules of GED for completing the table of distance are:

- The value of cell [э,э] is set to zero. In other words, Distance [1,1] = 0.
- Second row from cell [1, 2], must be filled with integers starting from one and incrementally increase the number by one for the cell next to it on the right. This goes on until the last cell is covered. Following this order results in facing the Distance [1, n+1] =n.

- Third column from cell [2, 1], must be filled with integers starting from one and incrementally increase the number by one for the cell at the bottom of it. This goes on until the last cell is covered. Following this order results in facing the Distance [m+1, 1] =m.

The table of distance looks like the Table 11 up to this point. It is it worth mentioning that cells containing integers in a grey color represent the nodes of the graphs and are not integers.

Table 10. Completing the second row and second column

	϶	1	1	2	1	7	7	5	1	7	2
϶	0	1	2	3	4	5	6	7	8	9	10
2	1										
1	2										
7	3										
5	4										
1	5										
2	6										

The rest of the cells must follow Equation 1:

$$i = index\ of\ a\ node\ from\ the\ source\ graph,$$

$$j = index\ of\ a\ node\ from\ the\ target\ graph,$$

$$X = \begin{cases} 0 & if\ \ source\ node\ of\ (i) = target\ nodeof\ (j) \\ 1 & if\ \ source\ node\ of\ (i) \neq target\ nodeof\ (j) \end{cases}$$

$$Distance(i,j) = min \begin{cases} Distance(i-1, j-1) + X \\ Distance(i-1, j) + 1 \\ Distance(i, j-1) + 1 \end{cases}$$

Equation 1. Main formula of table of distance

The formula above simply puts forth the fact that the value of each inner cell (red cell) depends on the value of three cells (green cells). As an example, we find the value of red cell in Table 12.

Table 11. Filling out the red cell using green cells

	϶	1	1	2	1	7	7	5	1	7	2
϶	0	1	2	3	4	5	6	7	8	9	10
2	1	🟥									
1	2										
7	3										
5	4										
1	5										
2	6										

$$i = 2, j = 2, source\ node\ of\ (2) = 1,\ \ target\ node\ of\ (2) = 2$$

$$source\_node(2) \neq target\_node(2), \text{so } X = 1$$

$$Distance(2{,}2) = min \begin{cases} Distance(1{,}1) + X = 0 + 1 = 1 \\ Distance(1{,}2) + 1 = 1 + 1 = 2 \\ Distance(2{,}1) + 1 = 1 + 1 = 2 \end{cases}$$

$$Distance(2{,}2) = 1$$

Equation 2. Example of evaluation of a cell in distance table

We do the same for another red cell.

Table 12. Filling out the red cell using green cells (2)

	϶	1	1	2	1	7	7	5	1	7	2
϶	0	1	2	3	4	5	6	7	8	9	10
2	1	1	2	2	3	4	5	6	7	8	9
1	2	1	1	2	2	3	4	5	6	7	8
7	3	2	2	2	3	2	🟥				
5	4	3	3	3	3	3					
1	5	4	3	4	3	4					
2	6	5	4	3	4	4					

$$i = 7, j = 4, source\ node\ of\ (7) = 7, target\ node\ of\ (4) = 7$$

$$source\ node\ of\ (7) = target\ node\ of\ (4), \text{so } X = 0,$$

$$Distance(7,4) = min \begin{cases} Distance(6,3) + X = 3 + 0 = 3 \\ Distance(6,4) + 1 = 2 + 1 = 3 \\ Distance(7,3) + 1 = 4 + 1 = 5 \end{cases}$$

$$Distance(7,4) = 3$$

Equation 3. Another example of evaluation of a cell in distance table

After finding the values of every cell, the green cell in Table 14, represents the final result. According to this example, by deleting four nodes (system calls) from the source graph, it will match the target graph.

Table 13. Completed table

	э	1	1	2	1	7	7	5	1	7	2
э	0	1	2	3	4	5	6	7	8	9	10
2	1	1	2	2	3	4	5	6	7	8	9
1	2	1	1	2	2	3	4	5	6	7	8
7	3	2	2	2	3	2	3	4	5	6	7
5	4	3	3	3	3	3	3	3	4	5	6
1	5	4	3	4	3	4	4	4	3	4	5
2	6	5	4	3	4	4	5	5	4	4	4

3.4.3.5. Output of the Testing Module (GED)

The fundamental responsibility of the testing module is to recognize the traces that are far away from the normal model built within the training phase. By using GED, we can find the minimum distance between any two traces of system calls. For the output of this module, we present the attack trace, the closest normal trace and the distance between them.

3.5. Content of ADUMAS

Although ADUMAS in this thesis is dealing with three agents, but it can be expanded to cover a bigger or smaller amount of agents depending on the desired methodologies. It is worth mentioning that the number of agents and modules depends on the number of approaches involved. The number of agents must be equal to the number of methodologies. Each agent will contain three modules and there will be one *data processing module* for the whole environment.

Hence, for a system covering n anomaly detection methods, there must be n agents and 3n +1 modules in total.

3.5.1. Modules

In this section, we cover the strategy of algorithms, starting from the *data processing module*. Since the updating modules function whenever the agents need to communicate, so the description of agent communication is placed in Chapter 4.

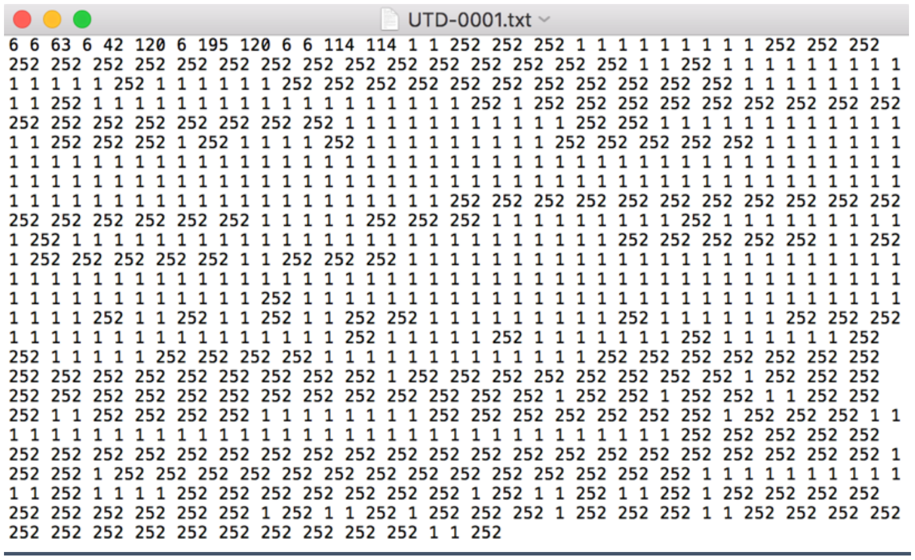
3.5.1.1. Data Processing Module

This module is in charge of dealing with the entry dataset to ADUMAS. System call based datasets can be fed into this module and the output will be a rearrangement of the whole dataset into sequences (traces) of system calls with the length greater than six system calls.

Generally, the datasets include three categories below.

- Training Traces: This category is composed of traces that are labeled normal.
- Validation Traces: This category is composed of traces that are mostly normal but it is not clear which ones are normal and which are abnormal.
- Attack Traces: This category is composed of attack traces.

Within each dataset there will be categories of training, validation and attack sequences. All categories will be turned into traces of system calls depending on the existing traces or files in the package. For instance, the ADFA-LD has 833 files in training package, so the output of this module will be 833 traces, each looking like Figure 10.



```
6 6 63 6 42 120 6 195 120 6 6 114 114 1 1 252 252 252 1 1 1 1 1 1 1 1 1 252 252 252
252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 1 1 252 1 1 1 1 1 1 1 1 1
1 1 1 1 1 252 1 1 1 1 1 1 252 252 252 252 252 252 252 252 252 252 252 1 1 1 1 1 1 1 1
1 1 252 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 252 1 252 252 252 252 252 252 252 252 252
252 252 252 252 252 252 252 252 1 1 1 1 1 1 1 1 1 1 1 252 252 1 1 1 1 1 1 1 1 1 1 1 1
1 1 252 252 252 1 252 1 1 1 1 252 1 1 1 1 1 1 1 1 1 252 252 252 252 252 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 252 252 252 252 252 252 252 252 252 252 252
252 252 252 252 252 252 1 1 1 1 1 252 252 252 1 1 1 1 1 1 1 1 1 252 1 1 1 1 1 1 1 1 1
1 252 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 252 252 252 252 252 1 1 252
1 252 252 252 252 252 1 1 252 252 252 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 252 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 252 1 1 252 1 1 252 1 1 252 252 1 1 1 1 1 1 1 1 1 252 1 1 1 1 1 1 252 252 252
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 252 1 1 1 1 1 252 1 1 1 1 1 1 1 252 1 1 1 1 1 1 252
252 1 1 1 1 1 252 252 252 252 1 1 1 1 1 1 1 1 1 1 1 1 1 252 252 252 252 252 252 252
252 252 252 252 252 252 252 252 252 1 252 252 252 252 252 252 252 252 1 252 252 252
252 252 252 252 252 252 252 252 252 252 252 252 252 1 252 252 1 252 252 1 1 252 252
252 1 1 252 252 252 252 1 1 1 1 1 1 1 1 252 252 252 252 252 252 252 1 252 252 252 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 252 252 252 252 252
252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 252 1
252 252 1 252 252 252 252 252 252 252 252 252 252 252 252 252 252 1 1 1 1 1 1 1 1 1 1
1 1 252 1 1 1 1 252 252 252 252 252 252 252 1 252 1 1 252 1 1 252 1 252 252 252 252
252 252 252 252 252 252 1 252 1 1 252 1 252 252 252 1 252 252 252 1 1 252 252 252 252
252 252 252 252 252 252 252 252 252 252 1 1 252
```

Figure 10. Sample of output of data processing module

3.5.1.2. Training Modules

The inputs to this module are training and validation traces. Depending on the methodology of the agent, we build a model based on normal traces.

- In the first agent, we break the traces into fixed length subsequences of system calls and build the model based on unique subsequences.
- In the second agent, we break the traces into fixed length subsequences of system calls, estimate the five features for each subsequence, and build a model with the range of normal values for each feature.
- In the third agent, we change each trace of system calls into a graph.

Once the above process is done, the output will be the entry point of the *testing modules*.

3.5.1.3. Testing Modules

Once we build the model based on normal and validation traces of system calls within the testing module for each attack trace, we find out how far it is from the normal model. In this way we find the anomalies and give a label to the whole trace.

- In the first agent we estimate the number of subsequences of system calls within the attack trace that match to the normal model. According to the calculations of the STIDE, we assign a weight to each trace.
- In the second agent, we estimate the five features for each attack trace which is the average of five features of the subsequences of system calls within itself. Afterwards, we find out how far it is from the normal range and based on the difference assign a weight to it.
- In the third agent, we find the closest normal graph to each attack graph and based on the difference we assign a weight to the attack trace.

3.5.2. Agents

According to the structure of ADUMAS, there are three agents involved, and each are deploying a unique methodology. This ends up analyzing the same dataset from three different points of view, and using the benefit of collaboration between these agents.

The fundamental role of the agents is in labeling the traces either N, referring to *normal* or A, referring to *anomaly*. They also assign a weight to traces alongside their labels, putting forth the level of trust regarding the detection.

Chapter 4 – Evaluation

The intention of using MAS is to make the approach more flexible in terms of the number of involved agents, the anomaly detection that are supported, as well as the costs considered for communication within the MAS.

In this chapter, we evaluate the results of individual agents, which will be the output of the *testing modules*, as well as the final result of ADUMAS. Once the output of all *testing modules* is ready, the third module will start operating. This module is responsible for making decisions according to the costs of communication between the agents.

For presenting the results of ADUMAS, we use the Receiver Operating Characteristic (ROC) curve [40]. This curve is a graphical plot, which represents the accuracy of detection while we deal with a binary classification, by using a threshold ranged from zero to 100%. This curve is created based on two parameters True Positive Rate (TPR) and False Positive Rate (FPR) [2].

TPR and FPR are part of the confusion matrix that is composed of four fundamental parameters [2]. In the confusion matrix the terms *positive* and *negative* refer to normal and abnormal, and while dealing with an accurate detection it is called *true* and inaccurate detection it is called *false*. The four parameters of a confusion matrix are explained in the following.

- True Positive (TP): This parameter relates to the data points (traces) that are normal and are accurately detected, meaning they are in fact normal and were labeled normal.

- True Negative (TN): This parameter relates to the data points that are abnormal and are accurately detected, meaning they are anomalous and were labeled anomalous.
- False Positive (FP): This parameter relates to the data points that are in fact normal but are inaccurately detected, meaning they are labeled as anomalies by mistake.
- False Negative (FN): This parameter relates to the data points that are normal but are inaccurately detected, meaning they are labeled normal by mistake.

We evaluate the parameters of confusion matrix in order to plot the ROC curve for each agent separately, as well as one final ROC to represent the final outcome of ADUMAS. Each agent deals with the assigned dataset in a unique way according to its methodology, but the commonality of all of the involved agents is in estimating the four parameters of confusion matrix.

While using ADFA-LD dataset, as it was mentioned before, there are three categories of traces, which are training, validation, and attack. Despite the ADFA-LD dataset, UNM does not have validation traces, so we use 25% of training as validation.

The existence of validation traces is important for the purpose of estimating the parameters of confusion matrix. We force each agent to go through three phases to find the TRP and FPR parameters. We define these phases based on the structure of the methodology assigned to the agent. Using the results of these phases, we are able to find TPR and FPR parameters in Equation 4:

- TPR is called the sensitivity parameter of the confusion matrix. In the context of machine learning, it mostly refers to the probability of detection.

- FPR is called the fall-out parameter of the confusion matrix. In the context of machine learning, it mostly refers to the probability of the false alarm.

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

Equation 4. TPR and FPR estimation

After plotting the ROC curve, we are estimate the area under the curve, also known as AUC [41] [42]. This parameter in this thesis, is meant to represent the estimation of accurate detection of anomalies. We use this parameter in order to compare the results of single agents versus multiple agents.

4.1. Evaluation of Individual Agents

The final decision of ADUMAS on the assigned dataset is based on considering the cost of communication and feedback of agents on the output of *testing modules*. Thus, we begin to analyze the results of each agent before focusing on the decision making process within the *updating modules*.

4.1.1. First Agent

According to the details of first agent explained in Chapter 3.4.1., we analyzed each trace based on the normal/abnormal status of the fixed-length subsequences within them.

- Phase one: This phase starts once the agent receives the output of the *data processing module*. In this phase, we build the model based on the training traces, and then within the *testing module*, we test the attack traces and assign the number of normal and abnormal subsequences for every attack trace. In other words, we try to see our progress in detecting the anomalies while the model is not built based on only normal traces of the dataset.

- Phase two: After completing the first phase, we look for validity of the model by using training traces for building the model and test it with the validation traces. Since both sets of traces are normal, this phase helps us to validate the model and find the TP and FP parameters.
- Phase three: The last phase for this agent is to build the model based on all normal traces, which are the combination of training and validation. In this phase, the *testing module* is working on the attack traces. Analyzing the results from phase one and phase three, result in estimating the TN and FN parameters.

One way to detect the anomalies is to build the model based on a combination of training and validation traces, and then test the attack traces. At this phase of testing, we require a threshold to decide on the label of each trace. This threshold is mainly for pointing at the weight assigned to each trace, according to the methodology of the agent.

In order to find the best combination of confusion matrix parameters that lead to a higher AUC, we need to estimate the best threshold on validation traces. In other words, by considering a threshold on validation traces, we set a learning rate for the model. This learning rate allows a portion of the validation traces to be used in the process of completing the model. Changing the learning rate will affect the TP and FP parameters. Next, while testing the attack traces, TN and FN parameters will change because the model works differently under various learning rates.

Regardless of the phase, in all of the experiments we estimate the number of normal and abnormal fixed length subsequences within the trace. Equation 5 represents a parameter called

Anomaly_Percentage attached to each trace, which is the number of normal subsequences divided by the total number of fixed length subsequences.

$$Anomaly\_Percentage = \frac{AbnormalCount}{AbnormalCount + NormalCount} \in [0,1]$$

Equation 5. Anomaly percentage parameter

Within phase two, both sets of traces are normal. In other words, we build a model out of one set of normal traces, and we test the model using the other set of normal traces. The main intention of this phase is to illustrate the difference between the training and validation traces. The influence of different learning rates within this phase is applied towards the amount of validation traces allowed to be used in the model.

According to the aforementioned formula, which leads to the *Anomaly_Percentage* parameter, the range of its values belong to (0,1). Hence, we try different learning rates starting from 0.1 and increase it by 0.1.

For each learning rate, we follow these steps:

- Step one: Use the phase two in order to estimate the values of TP and FP.
- Step two: Process phase one and phase three using the same threshold used in the previous step.
- Step three: Traces that gain the matching label during step two belong to the estimation of TN parameter, and traces ending up with different labels in step two belong to the estimation of FN parameter.

According to the range of the *Anomaly_Percentage* parameter, and our approach in finding the thresholds, we reached nine learning rates. Figure 11 represents the ROC curve of the first agent using different learning rates. The values next to learning rates are the AUCs of ROC using a unique learning rate.

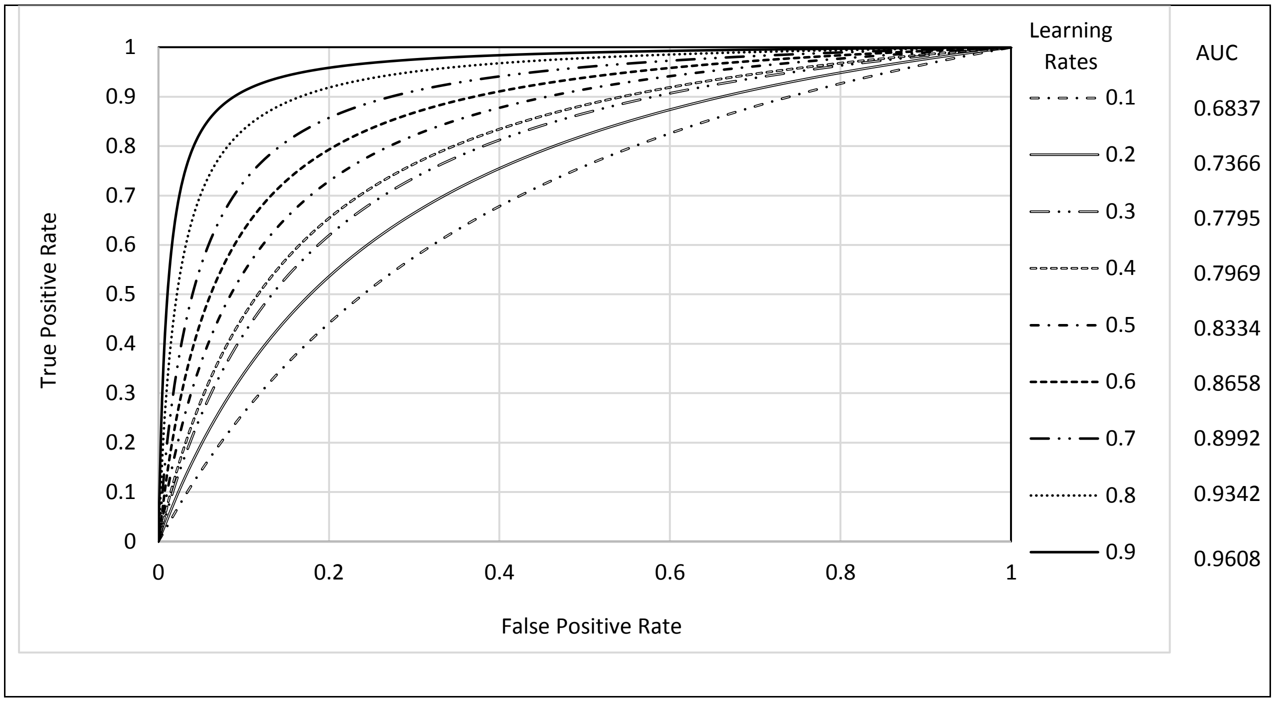


Figure 11. ROC curves of first agent using ADFA-LD, under different learning rates

Focusing on AUCs, by switching from learning rate 0.1 to 0.2, we witness an increase in the value of AUC compared to jumping from any other learning rate to its next learning rate. This refers to the fact that many of validation traces exist within the 0.2 distance of the training traces, and setting the threshold to this learning rate results in an increase in the number of validation traces to be used in building the model.

The goal of these examinations is to find a well-suited learning rate. We define a parameter called *Trend_Of_Validation_Usage* in order to decide on the learning rate that we want to assign to this agent.

$$Trend\_Of\_Validation\_Usage = Average(AUC, \frac{Used\_Validations\_in\_Model}{Total\ Validations})$$

Equation 6. Trend of validation usage parameter

This equation contains the effect of amount of the validation traces involved in the model, as well as the AUCs under this circumstance. As a result of using diverse learning rates, we encountered some spikes in this trend. We picked the closest learning rate next to the increase/decrease in the trend. It is worth mentioning we always ignore the extremes of the range as being thresholds. Since they are more affected by having 0% or 100% of validation traces, we do not rely on them.

In Figure 12 the x-axis represents a value in the range of [0,1], and y-axis represents the learning rates for this agent. The curves in this figure represent the AUCs of different learning rates using ADFA, the AUC difference of two adjacent learning rates, as well as the *Trend_Of_Validation_Usage* parameter. According to this figure, the first jump in learning rates from 0.1 to 0.2 holds more impact, with the AUC difference of 0.0529. The *Trend_Of_Validation_Usage* parameter contains its main spike in the range of [0, 0.2], and since we ignore the threshold of zero, we pick 0.2 to be assigned to the first agent as its main threshold. In the scenario where we deal with a MAS, we consider this learning rate for STIDE.

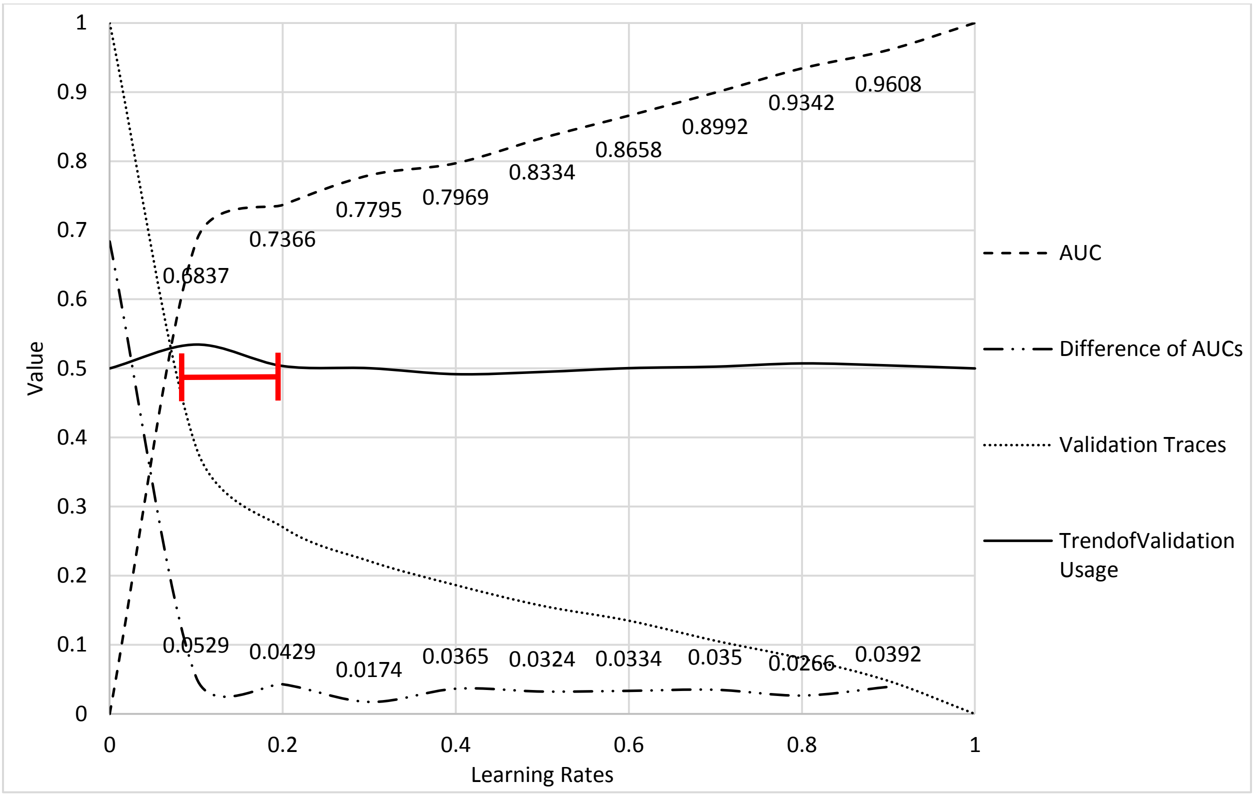


Figure 12. Trend of validation usage under different learning rates – STIDE – ADFA

We follow the same examination using UNM dataset. According to Equation 5, which leads to *Anomaly_Percentage* parameter, the range of its values belong to (0.1, 0.2). Hence, we'll try different learning rates starting from 0.12 and increase it by 0.02.

Figure 13 represents the ROC curve of the first agent using different learning rates. The values next to learning rates are the AUCs of ROC using a unique learning rate.

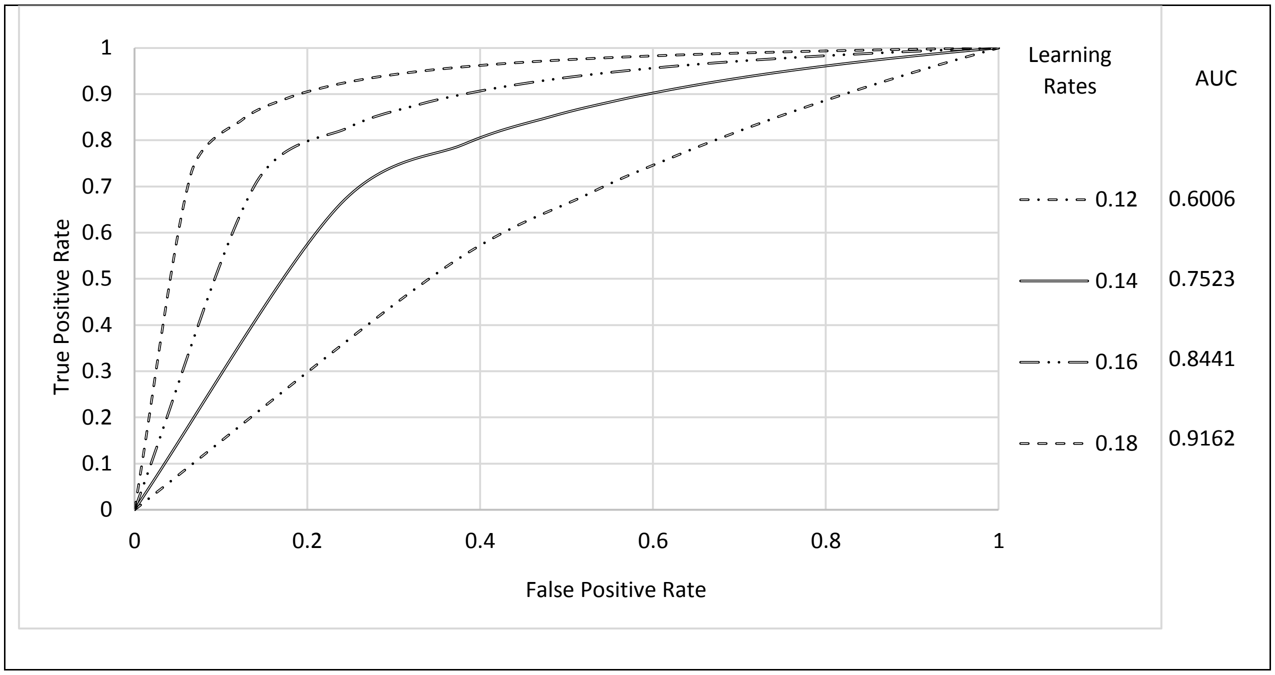


Figure 13. ROC curves of first agent using UNM, under different learning rates

Following the same methodology regarding the threshold we will assign to this agent, we look for the main spike, that is related to the biggest gap between the above ROC curves. The first agent using UNM dataset represents this gap close to a 0.12 learning rate, which we consider to be the threshold for this agent.

4.1.2. Second Agent

According to the details of the second agent explained in Chapter 3.4.2., we analyze each trace on the basis of their normal/abnormal status. In order to assign a weight to each trace, we define five parameters and estimate their values per trace. These weights lead to making better decisions for each trace.

The idea behind these parameters or features is to extract a meaning or pattern out of the traces. For instance, while analyzing normal traces, if we face system call 5 only appearing after system call 16, this can refer to the fact that the subsequence of 16-5 gives more normality weight to the trace.

These patterns or features all refer to the behavior of traces and we use values assigned to each feature in order to see whether it belongs to the normal range of that feature. For this purpose, we require having a normal range, in which we consider as the model for this agent. Similar to the first agent, we define three phases in order to examine the dataset and estimate the confusion matrix parameters.

- Phase one: This phase starts when the agent receives the output of the *data processing module*. In this phase, the agent is dealing with training traces. The goal of this phase is to estimate the range of normal traces for each feature.
- Phase two: In this phase, the agent is dealing with validation traces. Although these traces are also normal, but the range of values or more likely, the average of each feature, might be different from the average of phase one. In analyzing the results of this phase with the threshold set to the average of phase one, we end in an estimation of the TP and FP.
- Phase three: In the last phase of this agent, we deal with attack traces. After we estimate the features of each attack trace, we look for deviations from the normal range. This phase is covered in the *testing module*.

In the aforementioned phases, we mainly focus on finding the five features for each trace. But in order to label the trace, we need to follow these steps:

- Step one: We find the average of each trace.
- Step two: We use the results of step one and give a label of normal/abnormal to every feature.

- Step three: We merge the phase one and phase two, find a new average base on both sets of traces for each feature, and use it as a threshold for labeling the attack traces.

Once the validation and attack traces have five labels, we assign the number of abnormal features to each trace. Thus, we end up with a value between one and five according to the range of these features. The next step is to give a label to each trace, based on its abnormal features. Through this, we try different learning rates starting from two and increase it by one. Figure 14 represents the ROC curve of the second agent using different learning rates. The values next to learning rates are the AUCs of ROC using a unique learning rate.

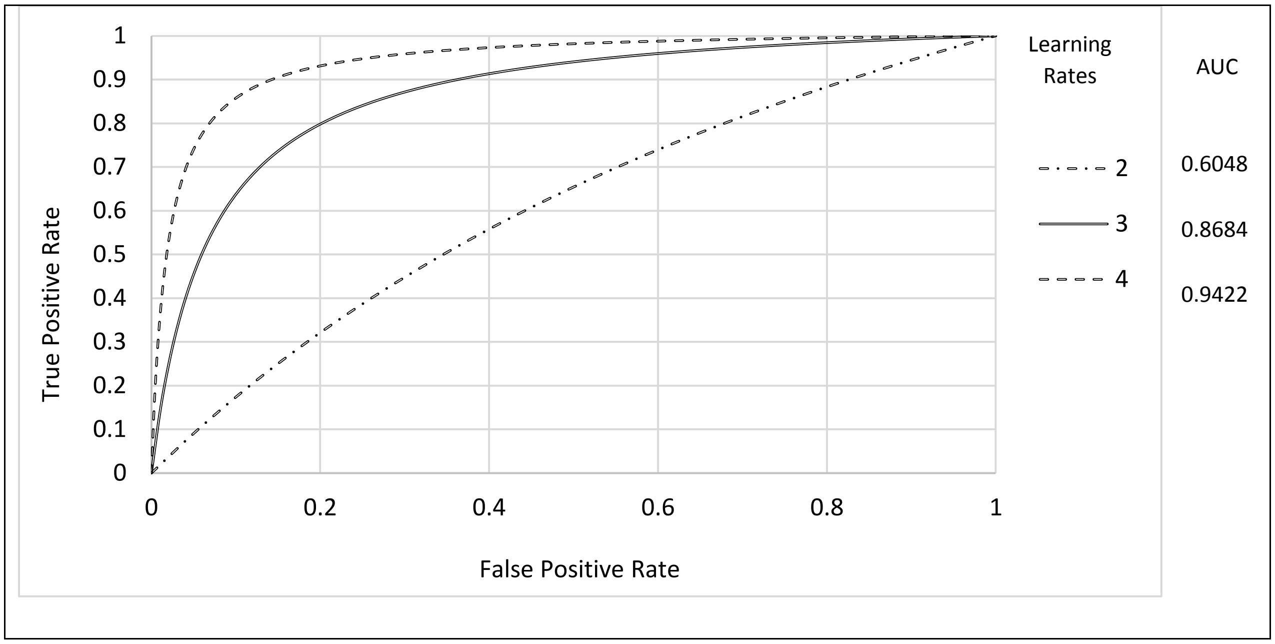


Figure 14. ROC curve of second agent using ADFA-LD, under different learning rate

Focusing on AUCs, by switching from learning rate 2 to 3, we witness an increase in the value of AUC compared to jumping from any other learning rate to its next learning rate. This refers to the fact that many validation traces exist within the 3 distance of the training traces, and setting the threshold to this learning rate, results in an increase in the number of validation traces to be used in building the model.

The target of these examinations is to find the most suited learning rate. As mentioned before, Equation 6 contains the effect of amount of validation traces involved in the model, as well as the AUCs under this circumstance. After analyzing the *Trend_Of_Validation_Usage* parameter, we pick the closest learning rate next to the main increase/decrease in the trend.

Figure 14 represents the AUCs of different learning rates using ADFA, the AUC difference of two adjacent learning rates, as well as the *Trend_Of_Validation_Usage* parameter. According to this figure, the first jump in learning rates from 2 to 3 holds more impact. The *Trend_Of_Validation_Usage* parameter contains its main decrease in the range of (2, 3), so we pick 3 to be assigned to the first agent as its main threshold. In the scenario where we deal with a MAS, we consider this learning rate for KNN.

In Figure 15 the x-axis represents a value in the range of [0,1], and y-axis represents the learning rates for this agent. The curves in this figure represent the AUCs of different learning rates using ADFA, the AUC difference of two adjacent learning rates, as well as the *Trend_Of_Validation_Usage* parameter. The *Trend_Of_Validation_Usage* parameter contains its main decrease in the range of (2, 3), and we pick 3 to be assigned to the second agent as its main threshold. In the scenario where we deal with a MAS, we consider this learning rate for KNN.

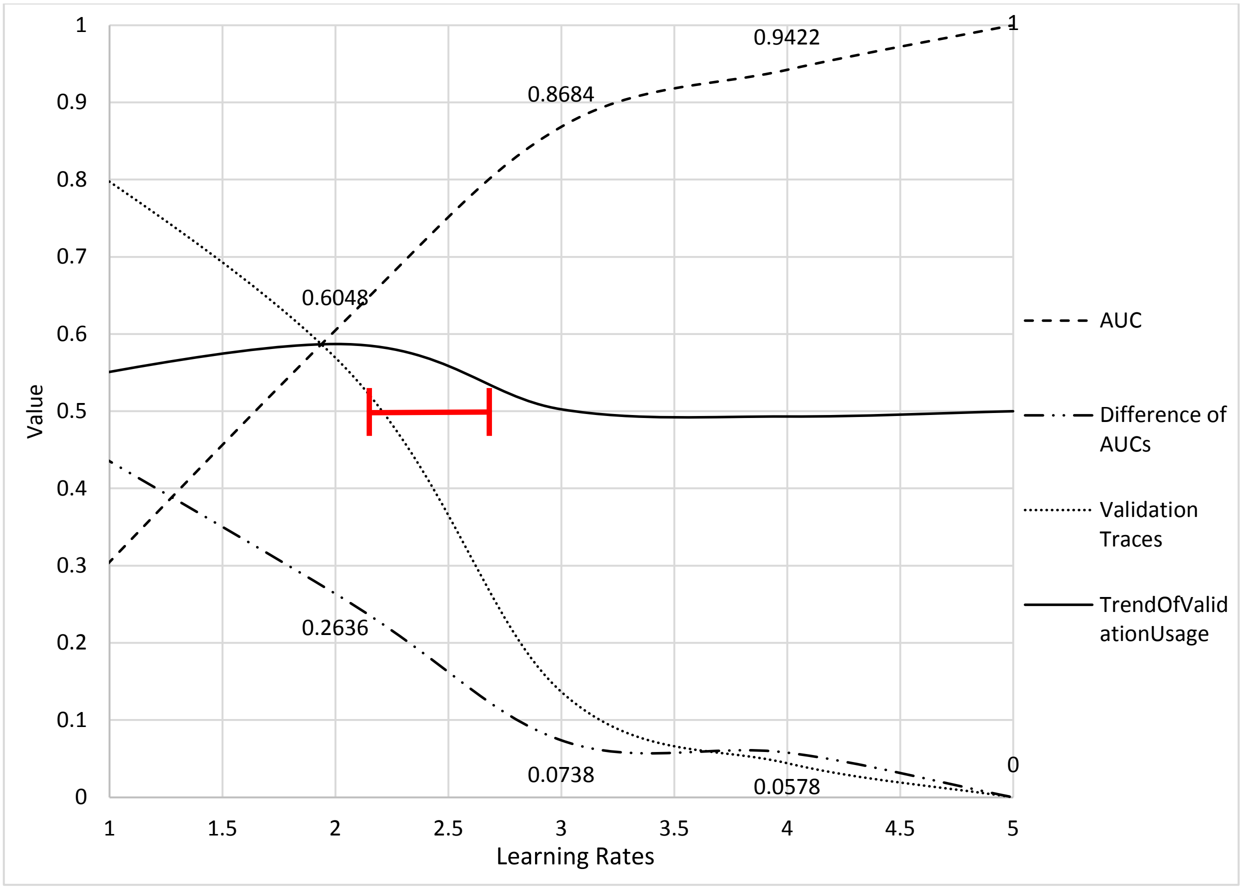


Figure 15. Trend of AUCs under different learning rates – KNN - ADFA

We follow the same examination using UNM dataset in Figure 16. After trying different learning rates, we set the threshold of this agent to 3.

Figure 16 represents the ROC curve of the second agent using different learning rates. The values next to learning rates are the AUCs of ROC using a unique learning rate.

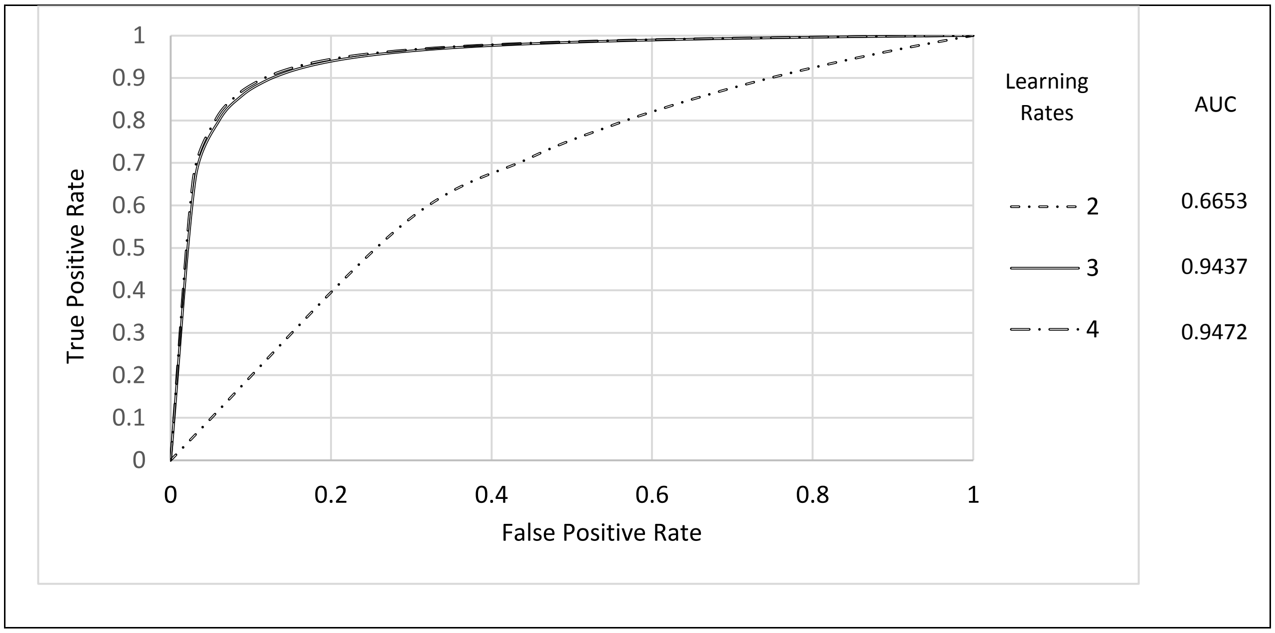


Figure 16. ROC curve of second agent using UNM, under different learning rate

Following the same methodology regarding the threshold we assign to this agent, we look for the main increase/decrease, which is related to the biggest gap between the above ROC curves. The first agent using UNM dataset represents this gap close to learning rate equal to 3, which we consider to be the threshold for this agent.

4.1.3. Third Agent

According to the details of the third agent, we find the minimum distance of two given traces. In both datasets that we use in this thesis, traces come from different lengths.

The same outcome of GED for different pairs of traces can infer different meanings. For instance, comparing two cases where the first pair of graphs owns the average size of 4000 and the outcome of GED is 4, to the case where the second pair of graphs owns the average size of

10, will highlight the fact that first pair is most likely normal, since they only have a minor difference.

Hence, the average size of involved traces becomes important, and since traces have different lengths, we normalize all of them. In the following figure, we go through the step by step process of getting final outcome of GED, considering the effect of size of traces.

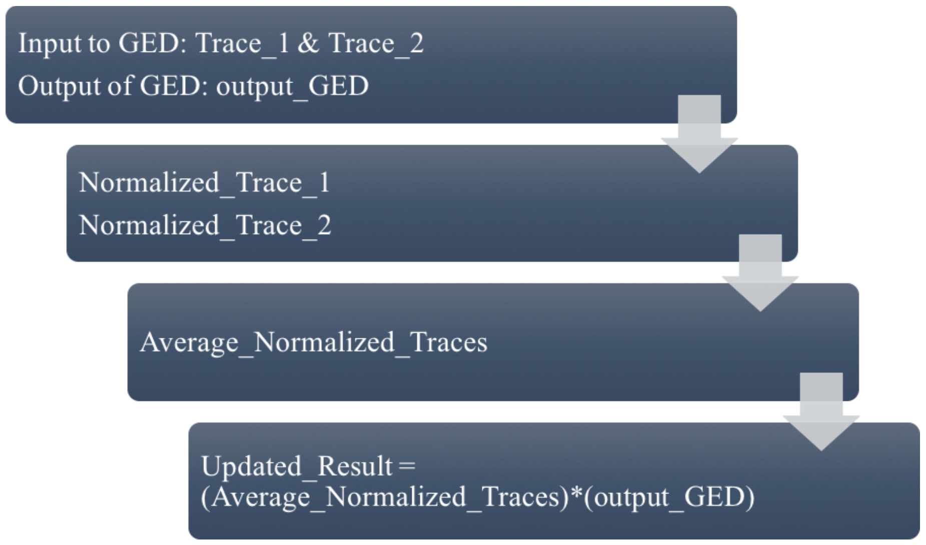

Input to GED: Trace_1 & Trace_2
Output of GED: output_GED
Normalized_Trace_1
Normalized_Trace_2
Average_Normalized_Traces
Updated_Result = (Average_Normalized_Traces)*(output_GED)

Figure 17. Estimating final output of GED

We through the aforementioned steps every time we use GED. Regardless of the dataset we are using, same as with the rest of the agents, we consider three phases for this agent as well.

- Phase one: This phase starts once the agent receives the output of *data processing module*, we estimate the minimum distance of all attack traces versus all training traces. Next, for each attack trace, we assign the closest training trace, the size of that trace, and after going through the aforementioned steps, the updated result of GED.
- Phase two: In this phase we estimate the minimum distance of all validation traces versus all training traces. Hence, for each validation trace, we assign the closest training trace,

the size of that trace, as well as the updated result of GED. In this experiment, we build the model based on training traces and we test the validation traces on the partially completed model. The results of this phase is useful in finding the TP and FP parameters.

- Phase three: In this phase we estimate the minimum distance of all attack traces versus the combination of all training and validation traces. Next, for each attack trace, we assign the closest normal trace, the size of that trace, and after going through the aforementioned steps, the updated result of GED.

According to the output of GED, the range of its values belong to (0,100). Hence, we try different learning rates starting from 10 and increasing it by ten. By using different learning rates, we allow the agent to use a specific portion of the normal traces to build the model.

Different learning rates lead to different confusion matrix parameters and, eventually, different ROC curves. Figure 18 represents the ROC curve of the third agent using different learning rates. The values next to learning rates are the AUCs of ROC using a unique learning rate.

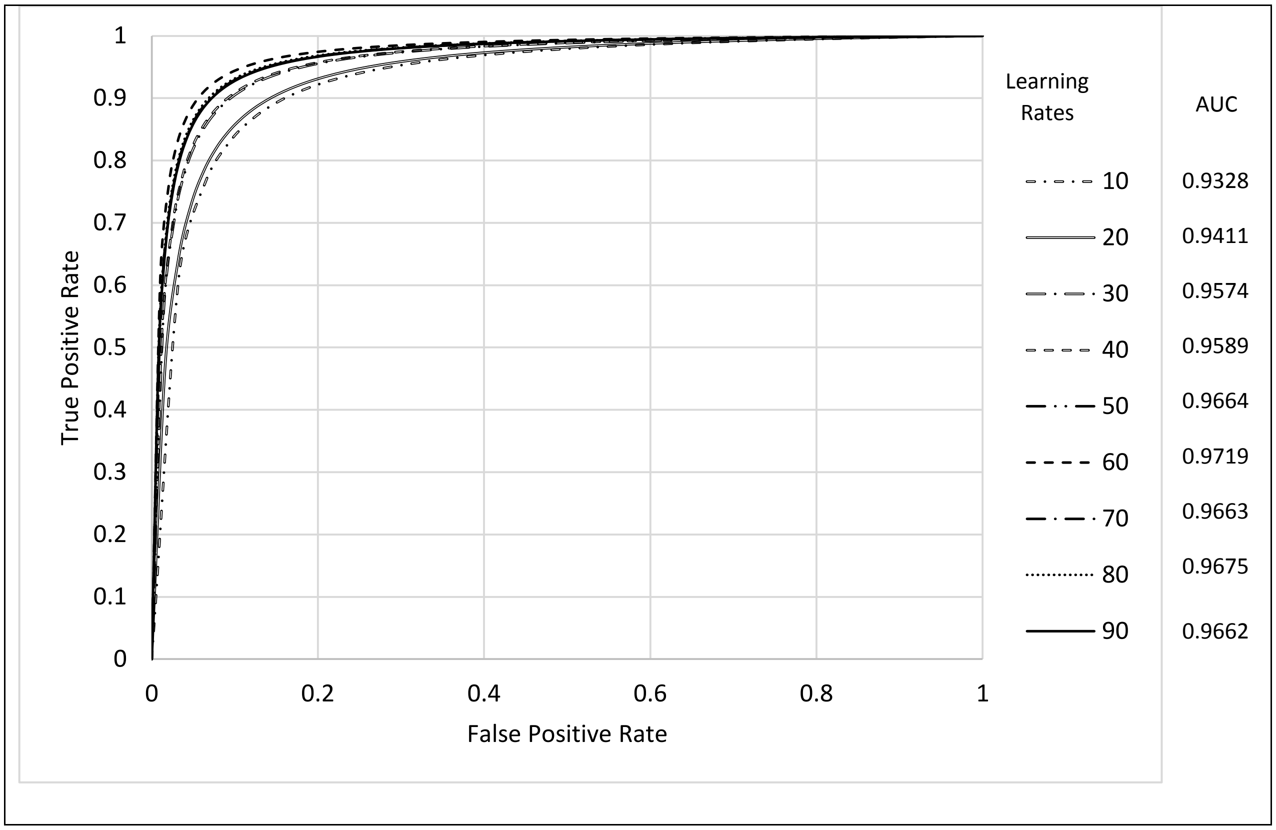


Figure 18. ROC curve of third agent using ADFA-LD, under different learning rate

Focusing on AUCs, by switching from learning rate 20 to 30, we witness an increase in the value of AUC compared to jumping from any other learning rate to its next learning rate. This refers to the fact that lots of validation traces exist within the 20 distance of the training traces, and setting the threshold to this learning rate results in an increase in number of validation traces to be used in building the model.

According to Equation 6, the *Trend_Of_Validation_Usage* parameter contains the effect of amount of validation traces involved in the model, as well as the AUCs under this circumstance. As a result of using diverse learning rates, we encounter some increase/decrease in this trend. We pick the closest learning rate adjacent to the main spike.

In Figure 19 the x-axis represents a value in the range of [0,1], and y-axis represents the learning rates for this agent. The curves in this figure represent the AUCs of different learning rates using ADFA, the AUC difference of two adjacent learning rates, as well as the *Trend_Of_Validation_Usage* parameter. According to this figure, the jump in learning rates from 10 to 20 holds more impact. The *Trend_Of_Validation_Usage* parameter contains its main spike in the range of (10, 20) and we pick 20 to be assigned to the first agent as its main threshold. In the scenario where we deal with a MAS, we consider this learning rate for GED.

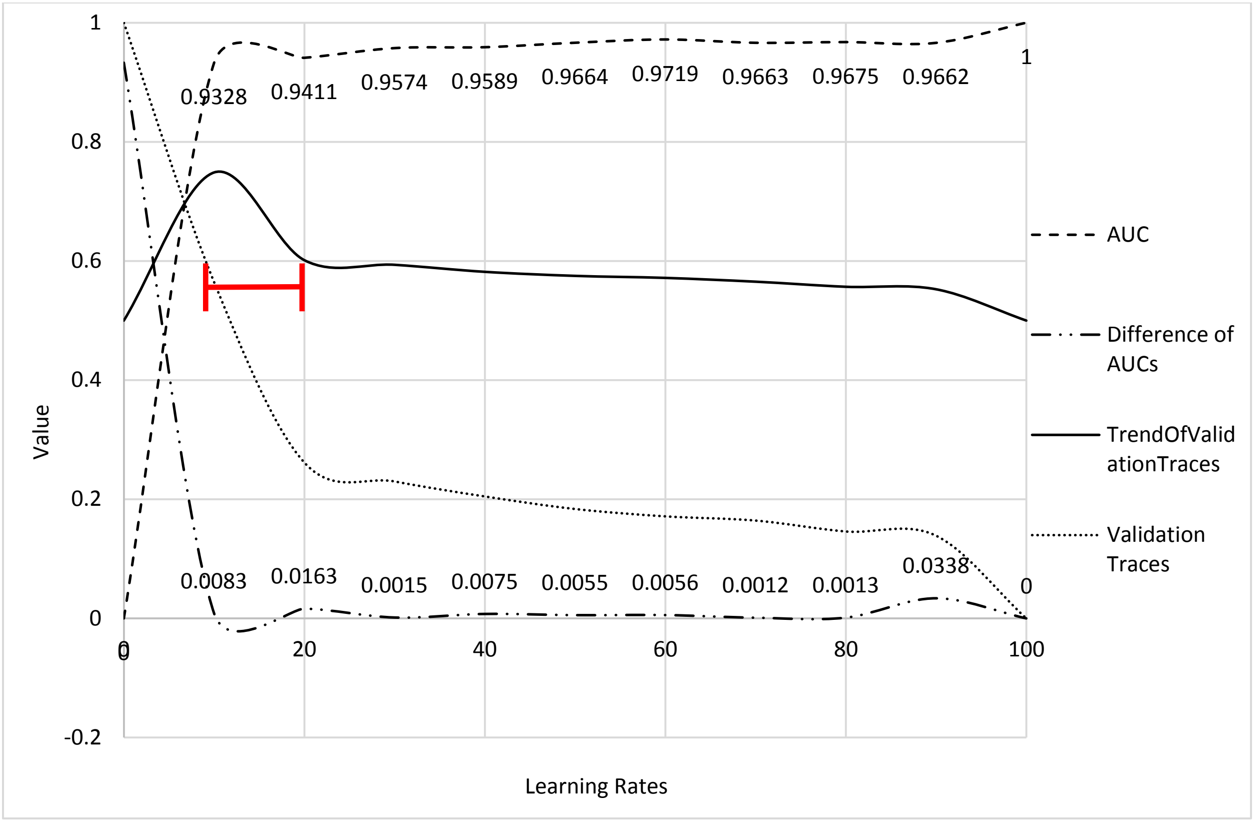


Figure 19. Trend of AUCs under different learning rates – GED - ADFA

We follow the same examination using UNM dataset. According to the results, the range of values belongs to (0, 600). Hence, we try different learning rates starting from 100 and increase it by 100. Figure 20 represents the ROC curve of the third agent using different learning rates. The values next to learning rates are the AUCs of ROC using a unique learning rate.

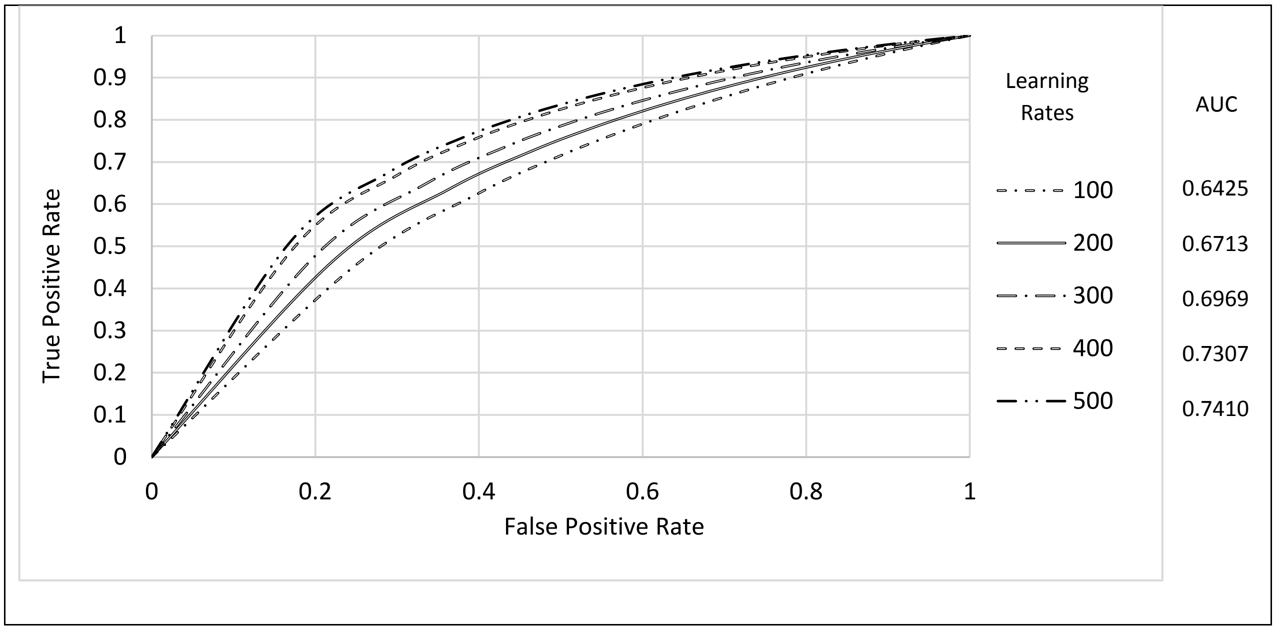


Figure 20. ROC curve of third agent using UNM, under different learning rate

Following the same methodology regarding the threshold that we assign to this agent, we look for the main spike, which is related to the biggest gap between the above ROC curves. The first agent using UNM dataset represents this gap close to a 100 learning rate, which we consider to be the threshold for this agent.

4.2. Evaluation of Multi-Agents

Once we estimate the results of each agent individually, we consider a cost for communication between the agents. This cost is represented as a weight that we assign to each attack trace depending on the methodology assigned to it.

For all of the agents, we assign a weight parameter, which defines the distance of the results from the threshold assigned to them. In the following section, we examine the results of different combinations of agents.

While dealing with two agents, we estimate the results for the combination of two agents, as well as the results of each agent individually. The intention of ADUMAS is to find out whether it should allow both agents to be involved in the decision making process, or trust only one of them.

In the following section, we image we deal with two agents and estimate their effect on ADUMAS.

4.2.1. STIDE-KNN

The first scenario for ADUMAS ignores the third agent and just analyzes the agents assigned with STIDE and KNN methodologies.

In the analysis of each agent separately, in the end we decided on a threshold, despite the fact that we faced an increase in AUC of both agents while using higher learning rates. It is worth mentioning again that the ROC curves for each agent under different learning rates were only for observing the behavior of dataset while using a portion of validation traces alongside the training traces for building the model. These analyses ended up deciding the threshold we assigned to each agent to be used in third phase, and these thresholds are 0.2 and 3, assigned to STIDE and KNN, respectively.

In this section, we focus on the communication costs between the agents and define the procedure for making decisions. The communication between the agents relies on the weight they assign to each trace with a specific methodology. In the MAS with two agents, there are two main scenarios:

- Both agents give the same label to each trace, so the decision of ADUMAS regarding this trace is clear.
- They are not on the same page regarding the labels they give to each trace. Within the process of estimating the TP and FP parameters, ADUMAS relies on the label with less weight. In dealing with TP and FP, we are interested in finding the normal traces. Since the weights refer to the anomalous level of the trace, less weight refers to more normality level. The opposite is with the TN and FN parameters. ADUMAS trusts the label assigned with more weight, since its obviously referring the abnormality level of the trace.

Despite the papers mentioned in the second chapter, we do not use a specific threshold for the communication cost. In ADUMAS we rely on the agent that holds more trust regarding the result it provides.

We start by using the final thresholds assigned to STIDE and KNN, follow the above steps, and discover the label of each trace. The result of ADUMAS for these two agents is obtaining the AUC of 0.8239, which is an increase from both individual agents. Figure 21 represents the cases that should be compared with each other. Since we have two agents, ADUMAS picks the better choice among these cases.

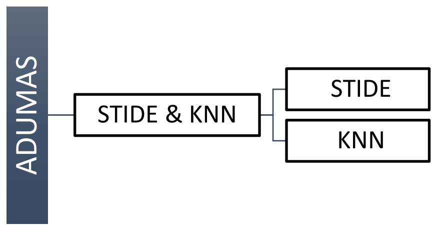


Figure 21. STIDE & KNN hierarchy

In order to be sure about these final thresholds, we increase the learning rates of the aforementioned agents and estimate the AUC of ADUMAS under the increased learning rates. Figure 22 represents the AUC of these cases:

- STIDE with its final threshold (0.2).
- KNN with its final threshold (3).
- Primary ADUMAS with final thresholds of STIDE and KNN.
- ADUMAS with increased learning rates in STIDE and KNN.

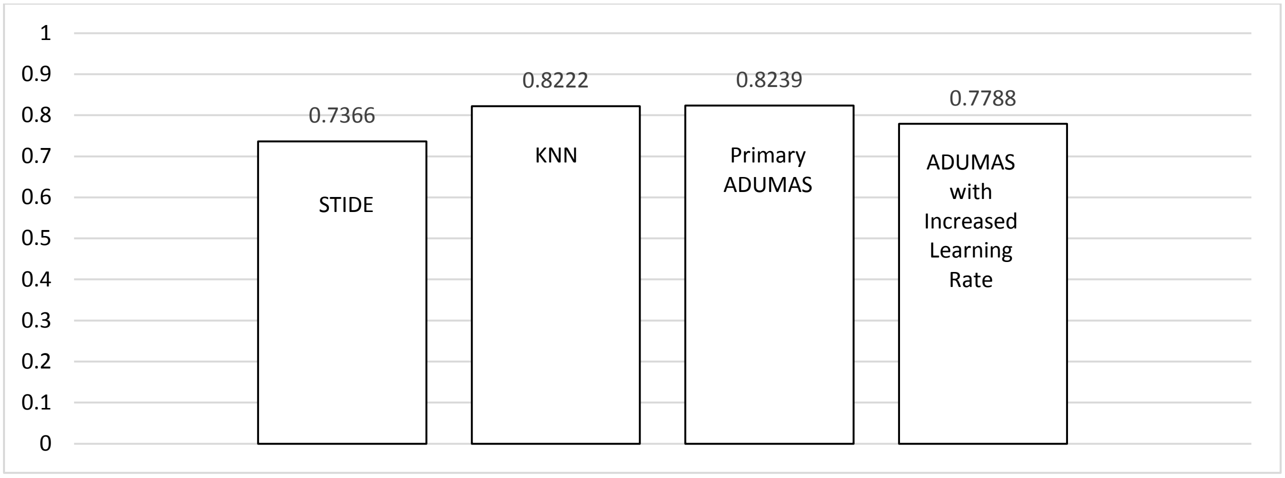


Figure 22. AUCs of four cases – STIDE & KNN - ADFA

By increasing the learning rates of STIDE and KNN, we obtain an ADUMAS with AUC of 0.7788. This refers to the fact that the thresholds assigned to STIDE and KNN are suit the system the best.

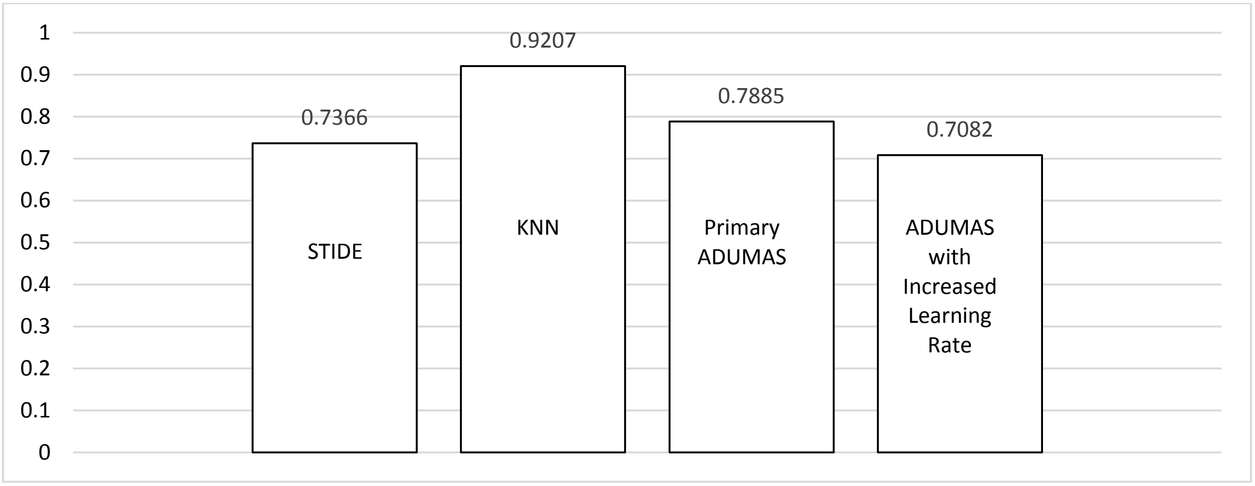


Figure 23. AUCs of four cases – STIDE & KNN - UNM

4.2.2. STIDE-GED

The second scenario for ADUMAS ignores the second agent and just analyzes the agents assigned with STIDE and GED. The thresholds we assigned for STIDE and GED are 0.2 and 10, respectively. Similar with ADUMAS dealing with STIDE and KNN, we focus on the communication costs between the agents and define the procedure for making decisions. The communication between the agents relies on the weight they assign to each trace with a specific methodology. In the MAS with two agents, there are two main scenarios:

- Both agents give the same label to each trace, so the decision of ADUMAS regarding this trace is clear.
- They are not on the same page regarding the labels they give to each trace. Within the process of estimating the TP and FP parameters, ADUMAS relies on the label with less

weight. Since weights refer to abnormality level of traces, ADUMAS relies on more weight while estimating TN and FN parameters.

In ADUMAS we rely on the agent that holds highest trust level, and we use the final thresholds of STIDE and GED, follow the above steps and find out the label of each trace. The result of ADUMAS for these two agents is obtaining the AUC of 0.7885, which is an increase for both individual agents. Figure 24 represents the cases that should be compared with each other. Since we have two agents, ADUMAS picks the better choice among these cases.

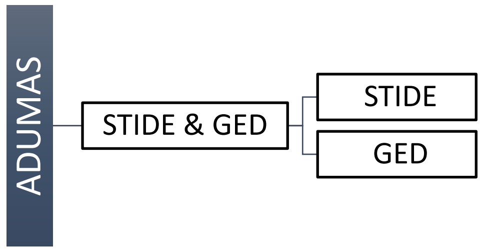


Figure 24. STIDE & GED hierarchy

In order to be certain about these final thresholds, we increase the learning rates of the aforementioned agents and estimate the AUC of ADUMAS under the increased learning rates. Figure 25 represents the AUC of these cases:

- STIDE with its final threshold (0.2).
- GED with its final threshold (10).
- Primary ADUMAS with final thresholds of STIDE and GED.
- ADUMAS with increased learning rates in STIDE and GED.

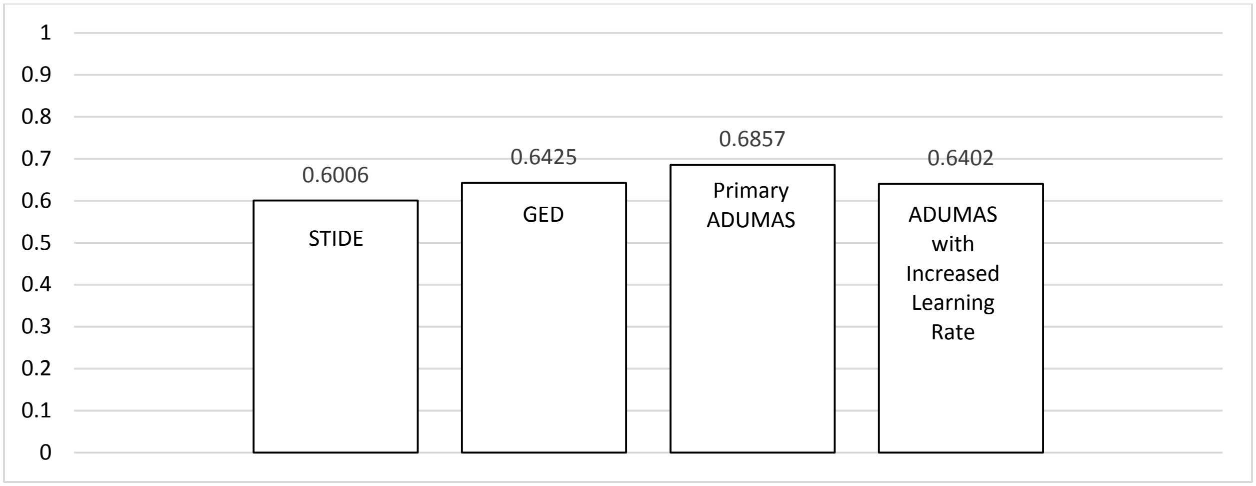


Figure 25. AUCs of four cases – STIDE & GED - ADFA

By increasing the learning rates of STIDE and GED, we obtain an ADUMAS with AUC of 0.7082. Hence, ADUMAS, in dealing with STIDE and GED, decides to go with GED under a threshold of 10. Figure 26 represents the results of UNM for this scenario.

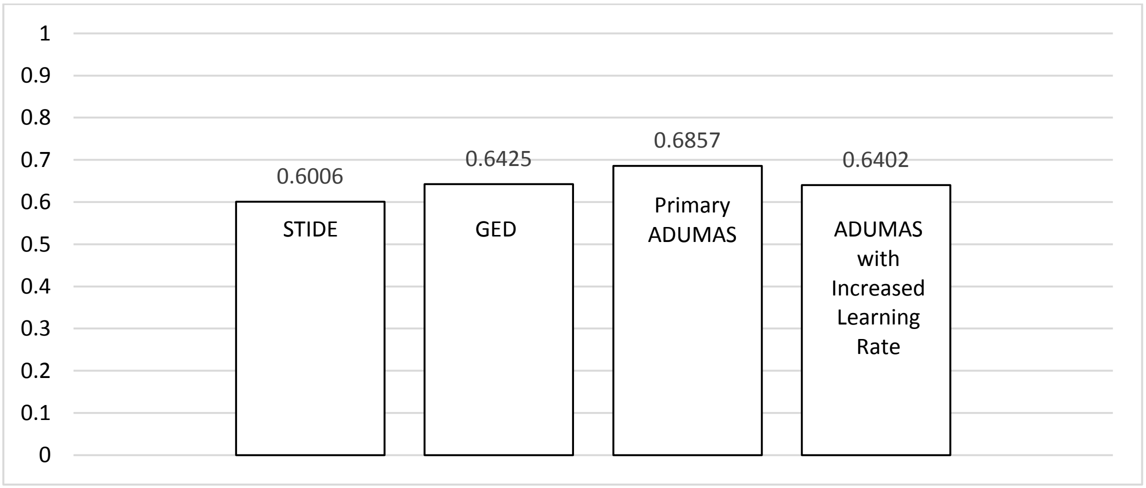


Figure 26. AUCs of four cases – STIDE & GED - UNM

By increasing the learning rates of STIDE and GED, we obtain an ADUMAS with AUC of 0.6857. Hence, ADUMAS, in dealing with STIDE and GED, decides to go with the MAS containing both agents.

4.2.3. KNN-GED

The third scenario for ADUMAS ignores the first agent and just analyzes the agents assigned with KNN and GED. The thresholds we assigned for KNN and GED are 3 and 10, respectively. In this scenario, we focus on the communication costs between the agents and define the procedure for making decisions. The communication between the agents relies on the weight they assign to each trace with a specific methodology. In the MAS with two agents, there are two main scenarios:

- Both agents give the same label to each trace. So the decision of ADUMAS regarding this trace is clear.
- They are not on the same page regarding the labels they give to each trace. Within the process of estimating the TP and FP parameters, ADUMAS relies on the label with less weight. Since weights refer to abnormality level of traces, ADUMAS relies on more weight while estimating TN and FN parameters.

In ADUMAS we rely on the agent that holds more trust level, we use the final thresholds of KNN and GED, follow the above steps and find out the label of each trace. The result of ADUMAS for these two agents is obtaining the AUC of 0.9707, which is an increase from both individual agents. Figure 27 represents the cases that should be compared with each other. Since we have two agents, ADUMAS picks the better choice among these cases.

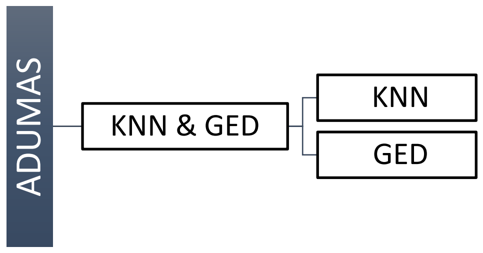


Figure 27. KNN & GED hierarchy

In order to make sure about these final thresholds, we increase the learning rates of the aforementioned agents, and estimate the AUC of ADUMAS under the increased learning rates. Figure 28 represents the AUC of these cases:

- KNN with its final threshold (3).
- GED with its final threshold (10).
- Primary ADUMAS with final thresholds of KNN and GED.
- ADUMAS with increased learning rates in KNN and GED.

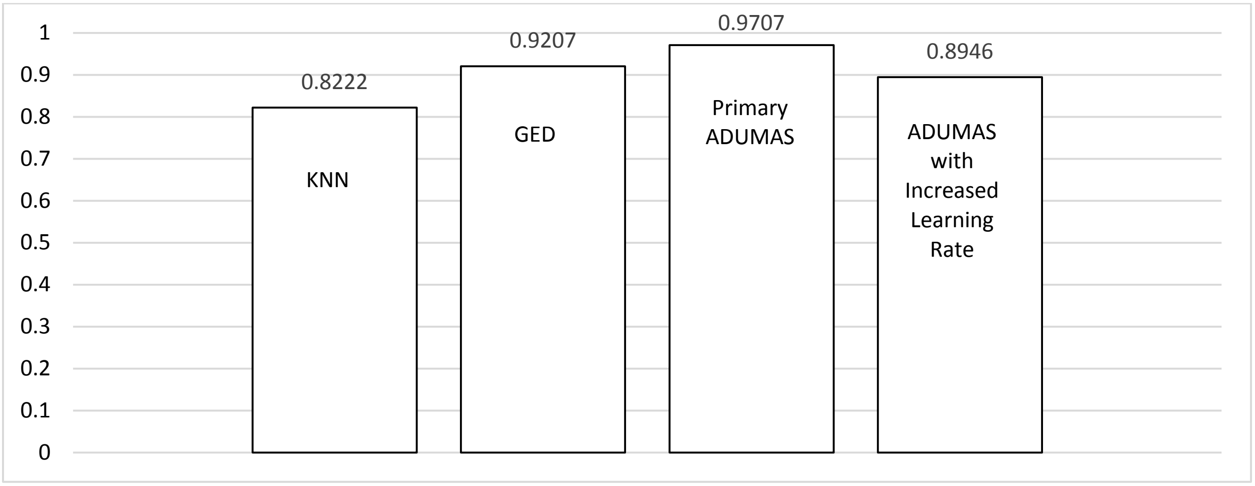


Figure 28. AUCs of four cases – KNN & GED - ADFA

By increasing the learning rates of KNN and GED, we obtain an ADUMAS with AUC of 0.9707. Hence, ADUMAS, in dealing with STIDE and GED, decides to go with combination of both agents with their final thresholds.

Figure 29 represents the results of this scenario using UNM dataset.

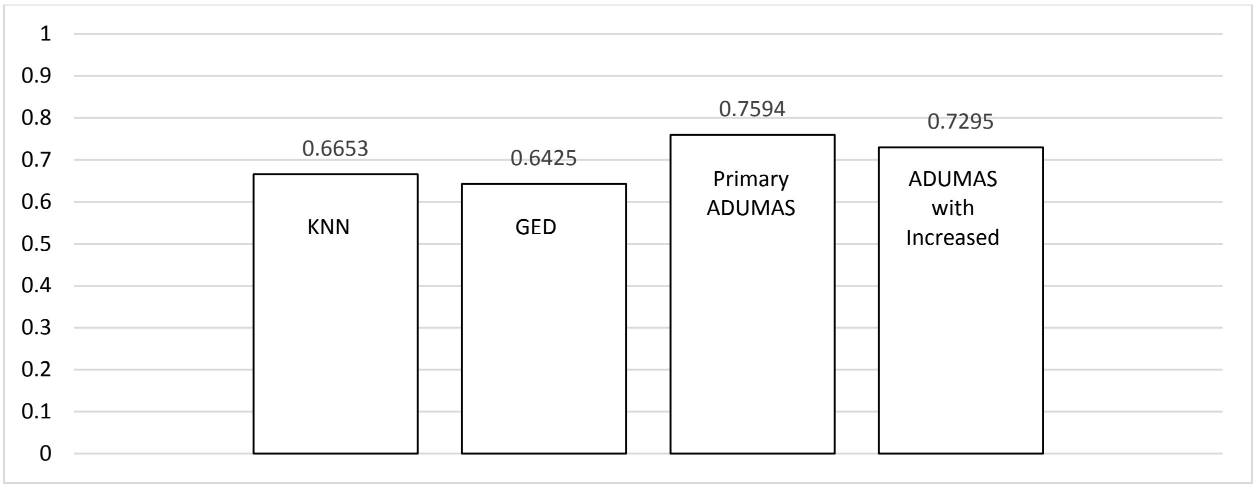


Figure 29. AUCs of four cases – KNN & GED - UNM

By increasing the learning rates of KNN and GED, we obtain an ADUMAS with AUC of 0.7594. Hence, ADUMAS, in dealing with STIDE and GED using UNM dataset, decides to go with combination of both agents with their final thresholds.

4.2.4. STIDE-KNN-GED

The last scenario for ADUMAS involves all three agents. The thresholds we assigned for STIDE, KNN, and GED are 0.2, 3, and 10, respectively. In this scenario, we focus on the communication costs between the agents and define the procedure for making decisions. The communication between the agents relies on the weight they assign to each trace within a specific methodology. In the MAS with three agents, there are two main scenarios:

- All three agents give the same label to each trace. So the decision of ADUMAS regarding this trace is clear.
- They are not on the same page regarding the labels they give to each trace. At this point, ADUMAS does not trust the label provided by two agents. These two might be assigned with weights close to threshold, and this represents a low level of trust. Within the

process of estimating the TP and FP parameters, ADUMAS relies on the label with less weight. Since weights refer to abnormality level of traces, ADUMAS relies on more weight while estimating TN and FN parameters.

In ADUMAS we rely on the agent(s) that hold a higher trust level, and we use the final thresholds of KNN and GED, and follow the above steps and ascertain the label of each trace. The result of ADUMAS for these three agents is obtaining the AUC of 0.9693. Figure 30 represents the cases that should be compared with each other. Since we have three agents, ADUMAS picks the better choice among cases of one agent, all combinations of two agents, and three agents.

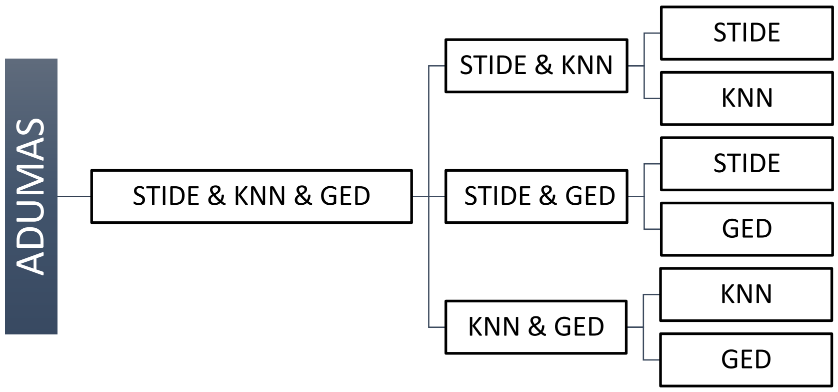


Figure 30. STIDE & KNN & GED hierarchy

In order to be certain about these final thresholds, we increase the learning rates of the aforementioned agents, and estimate the AUC of ADUMAS under the increased learning rates. Figure 31 represents the AUC of these cases:

- STIDE with its final threshold (0.2).
- KNN with its final threshold (3).
- GED with its final threshold (10).

- Primary ADUMAS with final thresholds of STIDE, KNN and GED.
- ADUMAS with increased learning rates in STIDE, KNN and GED.

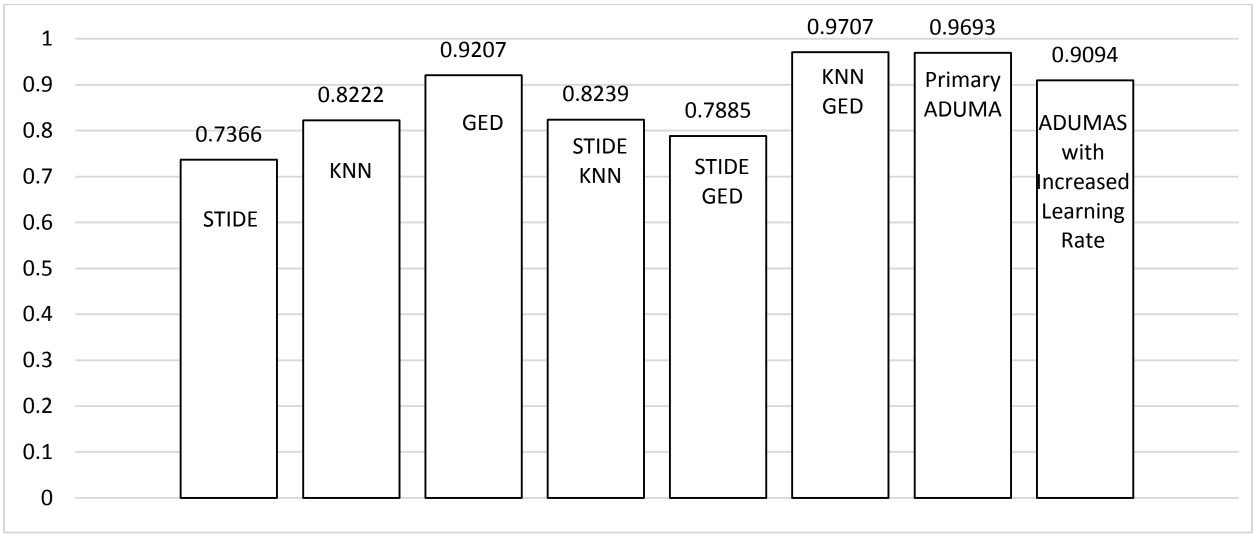


Figure 31. AUCs of eight cases – STIDE & KNN & GED - ADFA

Before increasing the learning rates, the primary ADUMAS reaches to AUC of 0.9707, which is the result that considers all the scenarios of one agents, two agents and all three agents. The best case is when ADUMAS ignores STIDE and relies on KNN and GED. By increasing the learning rates, ADUMAS fails to provide better detection. Hence, the final result of ADUMAS relates to second and third agents working with their final thresholds.

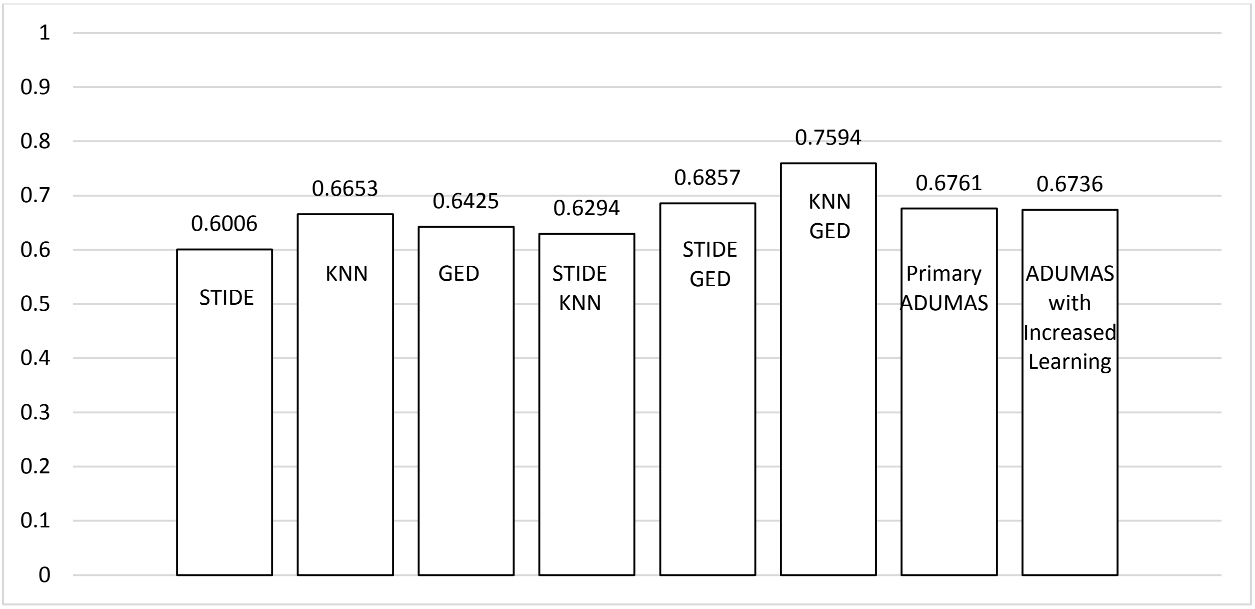


Figure 32. AUCs of eight cases – STIDE & KNN & GED - UNM

Before increasing the learning rates, the primary ADUMAS reaches to AUC of 0.7594, which is the result considering all scenarios of one agent, two agents and all three agents. The best case is when ADUMAS ignores STIDE and relies on KNN and GED. By increasing the learning rates, ADUMAS fails to provide better detection. Hence, the final result of ADUMAS using UNM relates to second and third agents working with their final thresholds.

Chapter 5 – Conclusion and Future Work

5.1. Conclusion

This thesis is concerned with studying the use of a MAS for anomaly detection purposes. We decided to use three agents, and assign one unique anomaly detection methodology to each agent. In the process of detecting anomalies, regardless of the algorithm we use, we need to build a model based on a normal dataset, so that later we can test the attack dataset on the built model. Since the nature of each methodology is different, we placed three modules inside each agent, dividing the task assigned to agent into three parts. The tasks assigned to these three modules are common between all agents, but the methodologies are different. The first module in each agent is responsible for training a normal model, the second module is responsible for testing the attack dataset on the built model, and the third module is responsible for considering the cost of communication and giving feedback to the rest of the agents.

We used a MAS in order to have a flexible platform in terms of communication and feedback between the agents, which helped to improve the accuracy of detection using more than one agent. This doesn’t necessarily mean that all the feedback from agents should be taken into account. Regardless of the number of involved agents in the system, if we look at them in a hierarchy, in order to reach to the top point where all the agents are involved in the decision making process, the layers below must be evaluated too. In other words, in a system with three agents, there are three subsystems dealing with two agents, and within those subsystems, there are two individual agents deploying a single methodology.

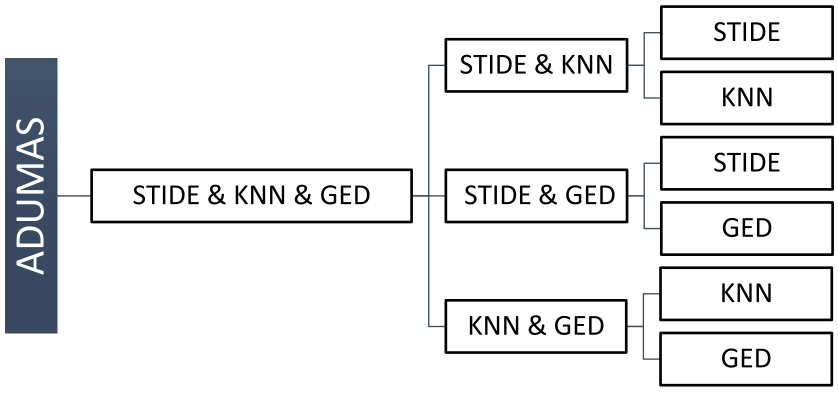


Figure 33. Hierarchy of MAS in ADUMAS

At each point, if the results do not show any improvement in the detection, we take a step back and reduce the number of agents involved. This refers to the fact that one of the agents was not providing helpful feedback to the rest of the agents. This depends on the datasets as well as the methodologies we use.

In Chapter 4, we examined the results of single agents as well as multiple agents under different learning rates. Depending on the learning rates, ADUMAS decided to go with the agent(s) performing better in terms of detection. The results are related to two different system call based datasets, ADFA-LD and UNM.

The structure of ADUMAS is not presenting the methodologies that act the best, but to present the better choices of agent(s) under specific conditions, among which are the datasets, or the learning rates. Hence, in different scenarios, ADUMAS will choose different combinations of agents, which leads to better results.

According to results of Chapter 4, in each scenario, the choice of ADUMAS is not the same under different datasets. This refers to the fact that datasets play an important role in decisions made by ADUMAS.

The main benefit of ADUMAS is in proving the existence of some keys that affect the results directly. These keys are:

- The methodologies being used. More precise and suitable methodologies for detection can result in better accuracy in regards to detection.
- The assigned thresholds for each methodology. Finding the right threshold has a great impact on the results.
- The selected features and weights for each methodology.

5.2. Future Work

The approach of this thesis can be expanded in many different directions.

5.2.1. Number of Agents

In this thesis we used three agents, but ADUMAS is flexible enough to work with many different numbers of agents. In terms of future work, the effect of more or less number of agents can be discussed, in order to see if using certain number of agents will complicate the statistics, especially if it happens to the point of a failure of the whole system.

5.2.2. Methodologies of Agents

Other future work regarding this thesis can be related to better choices of methodologies when assigning to agents. Since the dataset suitable for ADUMAS contains traces of system calls to examine the behaviors, the methodologies should be chosen in a way to properly handle this type of dataset.

5.2.3. Cost of Communication

There is more room to examine communication costs for agents in order to lead them towards better decision makings. There is room for making more precise decisions on the statistics of weight or level of trust regarding each methodology.

5.2.4. Learning Rates

In the process of building the model, since we deal with training traces as well as validation traces, we examined the effect of different learning rates on building the normal model. A future study can be done on investigating the learning rate that can achieve better results.

Bibliography:

[1] H. J. Liao, C. H. Richard Lin, Y. C. Lin, and K. Y. Tung, “Intrusion detection system: A comprehensive review,” *Journal of Network Computer Applications*, vol. 36(1), pp. 16–24, 2013.

[2] W. C. Lin, S. W. Ke, and C. F. Tsai, “CANN: An intrusion detection system based on combining cluster centers and nearest neighbors,” *Knowledge-Based Systems*, vol. 78(1), pp. 13–21, 2015.

[3] M. A. Faisal, Z. Aung, J. R. Williams, and A. Sanchez, “Data-stream-based intrusion detection system for advanced metering infrastructure in smart grid: A feasibility study,” *IEEE Systems Journal*, 9(1), pp. 31–44, 2015.

[4] 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,” *Elsevier Journal of Information Science*, vol. 378, pp. 484–497, 2017.

[5] L. Akoglu, H. Tong, and D. Koutra, “Graph based anomaly detection and description: A survey,” *Springer Journal of Data Mining and Knowledge Discovery*, 29(3), 2015.

[6] P. Garcia-Teodoro, J. Diaz-Verdejo, G. Macia-Fernandez, and E. Vazquez, “Anomaly-based network intrusion detection: Techniques, systems and challenges,” *Elsevier Journal of Computers and Security*, 28(1), pp. 18–28, 2009.

[7] W. Li and Q. Du, “Collaborative representation for hyperspectral anomaly detection,” *IEEE Transactions on Geoscience and Remote Sensing*, 53(3), pp. 1463–1474, 2015.

[8] G. Kim, S. Lee, and S. Kim, “A novel hybrid intrusion detection method integrating anomaly detection with misuse detection,” *Elsevier Journal of Expert Systems Applications*, 41(4) PART 2, pp. 1690–1700, 2014.

[9] W. Khreich, B. Khosravifar, A. Hamou-Lhadj, and C. Talhi, “An anomaly detection system based on variable N-gram features and one-class SVM,” *Elsevier Journal of Information Software Technology*, vol. 91, pp. 186–197, 2017.

[10] S. Aljawarneh, M. Aldwairi, and M. B. Yassein, “Anomaly-based intrusion detection system through feature selection analysis and building hybrid efficient model,” *Journal of Computational Science*, vol. 25, pp. 152–160, 2018.

[11] G. F. Cretu, M. E. Locasto, S. J. Stolfo, and A. D. Keromytis, “Casting out Demons: Sanitizing Training Data for Anomaly Sensors,” *Proceedings of IEEE Symposium on Security and Privacy*, , pp. 81-95, 2008.

[12] C. Gates and C. Taylor, “Challenging the Anomaly Detection Paradigm: A Provocative Discussion,” *Proceedings of the 2006 Workshop on New Security Paradigms*, pp. 21-29. 2006.

[13] R. Sommer and V. Paxson, “Outside the closed world: On using machine learning for network intrusion detection,” *Proceedings of IEEE Symposium on Security and Privacy* pp. 305-316, 2010.

[14] Y. Liao and V. R. Vemuri, “Use of k-nearest neighbor classifier for intrusion detection,” Elsevier Journal of *Computers and Security*, 21(5), pp. 439–448, 2002.

[15] D. Kang, D. Fuller, and V. Honavar, “Learning Classifiers for Misuse Detection Using,” *Proceedings of International Conference on Intelligence and Security Informatics*, pp. 511-516, 2005.

[16] W. T. Wong and S. H. Hsu, "Application of SVM and ANN for image retrieval," *Elsevier European Journal of Operational Research, 173(3),* pp. 938-950, 2006.

[17] S. Rawat, V. P. Gulati, A. K. Pujari, and V. R. Vemuri, "Intrusion Detection using Text Processing Techniques with a Binary-Weighted Cosine Metric," *Journal of Information Assurance and Security , Volume 1,* pp. 43–50, 2006.

[18] A. Sharma, A. K. Pujari, and K. K. Paliwal, "Intrusion detection using text processing techniques with a kernel based similarity measure," Elsevier Journal of *Computers and Security, 26(7-8),* pp. 488-495, 2007.

[19] E. De la Hoz, E. De La Hoz, A. Ortiz, J. Ortega, and B. Prieto, "PCA filtering and probabilistic SOM for network intrusion detection," *Elsevier Journal of Neurocomputing, Volume 164, pp. 71-81, 2015.*

[20] J. Bentahar and B. Khosravifar, "Using trustworthy and referee agents to secure multi-agent systems," *Proceedings of the 5th International Conference on Information Technology: New Generations,* pp. 477-482, 2008.

[21] M. Jacyno, S. Bullock, M. Luck, and T. R. Payne, "Emergent Service Provisioning and Demand Estimation through Self-Organizing Agent Communities Categories and Subject Descriptors," *Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, Volume 1,* pp. 481-488*, 2009.*

[22] C. Warrender, S. Forrest, and B. Pearlmutter, “Detecting intrusions using system calls: Alternative data models,” *Proceedings of IEEE Symposium on Security and Privacy* , pp. 133–145, 1999.

[23] S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff, “A sense of self for Unix processes,” *Proceedings of the IEEE Symposium on Security and Privacy,* pp. 120-128, 1996.

[24] S. Forrest, S. Hofmeyr, and A. Somayaji, “The evolution of system-call monitoring,” Proceedings of the *Annual Computer Security Applications Conference* , pp. 418-430, 2008.

[25] T. D. Huynh, N. R. Jennings, and N. R. Shadbolt, “An integrated trust and reputation model for open multi-agent systems,” *Springer Journal of Autonomous Agents and Multi-Agent Systems, 13(2), pp.* 119-154, 2006.

[26] H. S. Nwana, “Software agents: an overview,” *The Knowledge Engineering Review, 11(3),* pp. 205-244, 1996.

[27] W. Shen and D. H. Norrie, “Agent-Based Systems for Intelligent Manufacturing: A State-of-the-Art Survey,” *Springer Journal of Knowledge and Information Systems 1(2),* pp. 129-156, 1999.

[28] A. Parmar, J. Gnanadhas, T. T. Mini, G. Abhilash, and A. C. Biswal, “Multi-agent approach for anomaly detection in automation networks,” *Proceedings of Circuits, Communication, Control and Computing (I4C),* pp. 225-230, 2014.

[29] R. Becker, S. Zilberstein, V. Lesser, and C. V. Goldman, "Transition-Independent Decentralized Markov Decision Processes," *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multi-agent Systems,* pp. 41-48, 2003.

[30] E. Eskin, W. Lee, and S. J. Stolfo, "Modeling system calls for intrusion detection with dynamic window sizes," *Proceedings of DARPA Information Survivability Conference & Exposition II (DISCEX'01),* pp. 165-175, 2001.

[31] R. Becker, S. Zilberstein, V. Lesser, and C. V Goldman, "Solving Transition-Independent Decentralized Markov Decision Processes," *Journal of Artificial Intelligence Research, Volume 22,* pp. 423-455, 2004.

[32] P. Xuan, V. Lesser, and S. Zilberstein, "Communication decisions in multi-agent cooperation," *Proceedings of the5th International Conference on Autonomous Agents,* pp. 616-623, 2001.

[33] M. W. Allen, D. Hahn, and D. C. MacFarland, "Heuristics for multiagent reinforcement learning in decentralized decision problems," *IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL),* pp. 1-8, 2014.

[34] S. Jiang, G. Pang, M. Wu, and L. Kuang, "An improved K-nearest-neighbor algorithm for text categorization," *Elsevier Journal of Expert Systems with Applications, 39(1),* pp. 1503-1509, 2012.

[35] J. M. Keller and M. R. Gray, "A Fuzzy K-Nearest Neighbor Algorithm," *IEEE Transactions on Systems, Man, and Cybernetics, SMC-15(4),* pp. 580-585, 1985.

[36] X. Gao, B. Xiao, D. Tao, and X. Li, “A survey of graph edit distance,” *Springer Journal of Pattern Analysis and Applications 13(1),* pp. 113-129, 2010.

[37] https://www.cs.unm.edu/~immsec/systemcalls.htm

[38] https://www.unsw.adfa.edu.au/unsw-canberra-cyber/cybersecurity/ADFA-IDS-Datasets/

[39] D. K. S. Reddy and A. K. Pujari, “N-gram analysis for computer virus detection,” *Springer Journal of Computer Virology, 2(3),* pp 231-239, 2006.

[40] R. P. Lippmann *et al.*, “Evaluating intrusion detection systems without attacking your friends: The 1998 DARPA intrusion detection evaluation,” *Proceedings of DARPA Conference Exposition (DISCEX).* pp. 12–26, 1999.

[41] A. P. Bradley, “The use of the area under the ROC curve in the evaluation of machine learning algorithms,” *Elsevier Journal of Pattern Recognition, 30(7),* pp. 1145-1159, 1997.

[42] G. Creech and J. Hu, “A semantic approach to host-based intrusion detection systems using contiguous and discontiguous system call patterns,” *IEEE Transactions on Computers 63(4), pp. 807-819, 2014.*