

UNIVERSITÀ DELLA CALABRIA



**UNIVERSITA' DELLA CALABRIA**

Dipartimento di Ingegneria Informatica, Modellistica, Elettronica e Sistemistica

**Dottorato di Ricerca in**

Information and Communication Engineering for Pervasive Intelligent Environments

**CICLO**

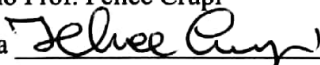
**XXIX**

**ENSEMBLE LEARNING TECHNIQUES  
FOR CYBER SECURITY APPLICATIONS**

**Settore Scientifico Disciplinare ING-INF/05**

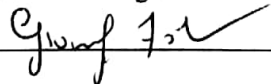
**Coordinatore:**

Ch.mo Prof. Felice Crupi

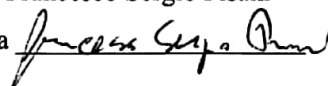
Firma 

**Supervisore:**

Ch.mo Prof. Gianluigi Folino

Firma 

**Dottorando:** Dott. Francesco Sergio Pisani

Firma 

Francesco Sergio Pisani

# Ensemble Learning techniques for Cyber Security Applications

November 9, 2017

To my family, for their love and their advice



---

## Acknowledgements

Firstly, I would like to express my sincere gratitude to my advisor Prof. Gianluigi Folino for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my Ph.D study.

Besides my advisor, I would like to thank all colleagues at ICAR and Unical for their insightful comments and encouragement, but also for the stimulating discussions and for all the fun we have had in the last three years.

I thank my friends for the sleepless nights, for the wonderful journeys we had and for all these years of fun.

Last but not the least, I would like to thank my family: my parents and my sister for supporting me throughout writing this thesis and my life in general.



---

## Abstract (English)

Cyber security involves protecting information and systems from major cyber threats; frequently, some high-level techniques, such as for instance data mining techniques, are used to efficiently fight, alleviate the effect or to prevent the action of the cybercriminals.

In particular, classification can be efficiently used for many cyber security applications, i.e. in intrusion detection systems, in the analysis of the user behavior, risk and attack analysis, etc.

However, the complexity and the diversity of modern systems opened a wide range of new issues difficult to address.

In fact, security softwares have to deal with missing data, privacy limitation and heterogeneous sources. Therefore, it would be really unlikely a single classification algorithm will perform well for all the types of data, especially in presence of changes and with constraints of real time and scalability.

To this aim, this thesis proposes a framework based on the ensemble paradigm to cope with these problems. Ensemble is a learning paradigm where multiple learners are trained for the same task by a learning algorithm, and the predictions of the learners are combined for dealing with new unseen instances. The ensemble method helps to reduce the variance of the error, the bias, and the dependence from a single dataset; furthermore, it can be built in an incremental way and it is apt to distributed implementations. It is also particularly suitable for distributed intrusion detection, because it permits to build a network profile by combining different classifiers that together provide complementary information. However, the phase of building of the ensemble could be computationally expensive as when new data arrives, it is necessary to restart the training phase. For this reason, the framework is based on Genetic Programming to evolve a function for combining the classifiers composing the ensemble, having some attractive characteristics. First, the models composing the ensemble can be trained only on a portion of the training set, and then they can be combined and used without any extra phase of training. Moreover the models can be specialized for a single class and they can be designed to handle the difficult problems of unbalanced classes and missing data.

In case of changes in the data, the function can be recomputed in an incrementally way, with a moderate computational effort and, in a streaming environment, drift strategies can be used to update the models. In addition, all the phases of the algorithm are distributed and can exploits the advantages of running on parallel/distributed architectures to cope with real time constraints.

The framework is oriented and specialized towards cyber security applications. For this reason, the algorithm is designed to work with missing data, unbalanced classes, models specialized on some tasks and model working with streaming data.

Two typical scenarios in the cyber security domain are provided and some experiment are conducted on artificial and real datasets to test the effectiveness of the approach. The first scenario deals with user behavior. The actions taken by users could lead to data breaches and the damages could have a very high cost. The second scenario deals with intrusion detection system. In this research area, the ensemble paradigm is a very new technique and the researcher must completely understand the advantages of this solution.



---

## Abstract (Italian)

La sicurezza informatica riguarda la protezione delle informazioni e dei sistemi dalle principali minacce informatiche; spesso, alcune tecniche di alto livello, come ad esempio le tecniche di data mining, vengono utilizzate per combattere e/o impedire l'azione dei criminali informatici.

In particolare, la classificazione può essere utilizzata in modo efficiente per molte applicazioni di sicurezza informatica, ad esempio, nei sistemi di rilevamento delle intrusioni, nell'analisi del comportamento degli utenti, nell'analisi del rischio e degli attacchi, ecc.

Tuttavia, la complessità e la diversità dei moderni sistemi ha creato nuove questioni difficili da affrontare.

Infatti, i software di sicurezza devono affrontare problemi di dati mancanti, di limiti imposti dalla privacy e di dati provenienti da fonti eterogenee. Pertanto, è molto improbabile che un singolo algoritmo di classificazione possa essere utilizzato per tutti i tipi di dati, soprattutto in presenza di cambiamenti e con vincoli di tempo reale e scalabilità.

Per tale scopo, questa tesi propone un framework basato sul paradigma degli ensemble per affrontare questi problemi. L'ensemble è un paradigma di apprendimento in cui più learner sono addestrati per lo stesso compito da un algoritmo di apprendimento, e le previsioni dei classificatori sono combinate per affrontare le nuove istanze. Il metodo degli ensemble aiuta a ridurre la varianza dell'errore, la polarizzazione, e la dipendenza da un singolo insieme di dati; inoltre può essere costruito in maniera incrementale ed è adatto per implementazioni distribuite. Inoltre, è particolarmente indicato per il rilevamento delle intrusioni perché permette di costruire un profilo di rete mediante la combinazione di diversi classificatori che, insieme, forniscono informazioni complementari. Tuttavia, la fase di costruzione dell'ensemble potrebbe essere computazionalmente costosa ed è necessario riavviare la fase di addestramento. Per questo motivo, il framework utilizza la programmazione genetica per evolvere una funzione di combinazione dei classificatori che compongono l'ensemble con alcune caratteristiche interessanti. Innanzitutto, i modelli possono essere addestrati solo su una parte del training set, e quindi possono essere combinati e utilizzati senza alcuna fase extra di training. Inoltre i modelli possono essere

specializzati per una singola classe e possono essere progettati per gestire i problemi di classi sbilanciate e dati mancanti.

L'analisi dei cambiamenti nel flusso dei dati necessita l'aggiornamento dei modelli. Per questo scopo la funzione di combinazione può essere ricalcolata in modo incrementale, con uno sforzo computazionale moderato e, considerando un ambiente streaming, l'attivazione della procedura di aggiornamento può essere determinata da algoritmi di drift detection. In aggiunta, tutte le fasi dell'algoritmo sono distribuite e possono sfruttare i vantaggi dell'esecuzione su architetture parallele e/o distribuite per far fronte a vincoli di tempo reale.

Il framework è orientato e specializzato per le applicazioni di cyber security. Per questo motivo, l'algoritmo è stato progettato per funzionare con dati mancanti, classi sbilanciate, modelli specializzati e modelli che lavorano su stream di dati.

Infine, sono stati presentati due scenari tipici del settore della cyber security e sono stati condotti degli esperimenti su dataset artificiali e reali testando l'efficacia del metodo. Il primo scenario si occupa della rilevazione del comportamento degli utenti. Le azioni intraprese dagli utenti potrebbero portare a violazioni di dati e i danni arrecati potrebbero avere un costo molto elevato. Il secondo scenario si occupa del rilevamento delle intrusioni. In questo settore, il paradigma degli ensemble è una tecnica nuova ed i ricercatori devono ancora comprendere appieno tutti i vantaggi di questa soluzione.

---

## Contents

<b>1</b>	<b>Introduction</b> .....	1
1.1	Thesis Overview .....	3
1.2	Selected and relevant publications .....	4
<b>2</b>	<b>Cyber security applications</b> .....	5
2.1	Users profile classification .....	6
2.1.1	Methods for classifying user profiles .....	7
2.2	An introduction to intrusion detection systems .....	8
2.2.1	The problems of modern NIDS architecture .....	8
2.2.2	NIDS environment and ensemble-based techniques .....	10
<b>3</b>	<b>Background</b> .....	11
3.1	Evolutionary algorithms and Genetic Programming .....	11
3.1.1	Parallel implementation of a GP tool .....	12
3.2	The problem of missing data .....	13
3.3	Ensemble-based techniques .....	16
3.4	Combining ensemble of classifiers .....	17
3.4.1	Ensemble of classifiers and non-trainable functions .....	17
<b>4</b>	<b>Related works</b> .....	21
4.1	Works on incomplete datasets and related approaches .....	21
4.1.1	Incomplete datasets and missing data .....	21
4.2	Ensemble and evolutionary algorithms for classification .....	23
4.3	Ensemble technique for cyber security applications .....	24
4.3.1	Data stream and time window .....	24
4.3.2	Ensemble technique and IDS systems .....	25
4.4	IDS techniques in distributed environment .....	27
4.4.1	Distributed and collaborative IDS .....	28
4.4.2	Collaborative IDS .....	28
4.4.3	Data mining-based approaches .....	31
4.4.4	High-performance implementations .....	34

<b>5</b>	<b>An architecture for combining ensemble of classifiers</b> .....	37
5.1	The technique for combining the ensemble of classifiers .....	38
5.1.1	The combining function .....	38
5.1.2	A distributed tool to evolve the combiner functions .....	39
5.2	The missing data problem and the framework solution .....	41
5.2.1	An architecture for classification of user profiles in e-payment systems .....	41
5.2.2	The software architecture of the meta-ensemble approach. . .	43
5.3	A distributed framework for intrusion detection .....	45
5.3.1	Preprocessing methodology .....	45
5.3.2	The architecture of the IDS .....	45
5.4	A meta-ensemble approach for combining specialized ensemble of evolved classifiers .....	48
5.5	The framework and the drift algorithms .....	50
5.5.1	Drift detection and replacement strategies .....	53
<b>6</b>	<b>Experimental results</b> .....	55
6.1	Description of the datasets used in the experiments .....	55
6.2	Experimental section on unbalanced data .....	59
6.2.1	Environment configuration and parameter settings .....	59
6.2.2	Comparing with other evolutionary strategies and ensemble techniques .....	60
6.2.3	Results for the KDDCup'99 dataset .....	62
6.3	Experimental results on missing features .....	64
6.3.1	Parameter settings .....	64
6.3.2	Experiments on the capacity of handling missing features . .	64
6.3.3	Analysis on a real-world dataset: the Unix dataset. ....	66
6.4	Experimental investigation on specialized ensemble approach . . . . .	69
6.4.1	Experiments on KDD 99 and ISCX IDS.....	69
6.5	Results with drift detection.....	72
6.5.1	Parameter settings .....	72
6.5.2	Experiments conducted on the artificial datasets.....	72
6.5.3	Experiments conducted on the real dataset .....	73
	<b>Conclusion and future works</b> .....	77
	<b>References</b> .....	81

## **Introduction**

In the last few years, the interest in cyber security problems is really increasing, as cyber crime seriously threatens national governments and the economy of many industries [22]. The complexity and the diversity of modern systems make addressing new issues more complex and the security has gained a growing importance.

Regarding this topic, the two main common questions are “how to protect a system” and “how to protect the users of a system”. The first issue is referred to technologies and policies to protect a system. The second one is related to protection of the users and to the prevention of malicious activities mainly caused by account violations. Obviously, the protection against external and malicious intrusion is the main priority of all the system administrators. Moreover, the damages caused from identity/credential theft are growing thanks to social engineering techniques and phishing attacks.

Many types of systems in the industry, in communication and in cyber-physical systems have security issues to handle. Due to complexity, a general/unique solution is not easy to find, but the problem must be solved by multiple points of view. For instance, the data mining techniques can be seen as a good solution for a particular class of problems. In fact, the classification can be efficiently used for many cyber security application, i.e. in intrusion detection systems, in the analysis of the user behavior, risk and attack analysis, etc. However, in this particular domain, datasets often have different number of features and each attribute could have different importance and costs. Furthermore, the entire system must also work if some datasets are not present. Therefore, it would be really unlikely a single classification algorithm will perform well for all the datasets, especially in presence of changes and with constraints of real time and scalability.

User profile classification and intrusion detection systems are common scenarios that can be coped with the techniques presented in this work.

The first scenario is a system, which permits to divide users in groups with same behavior and same weakness. This task is very important in modern architecture because the complexity makes difficult to detect attack started from legitimate users. Many data breaches come from user accounts and they are used as vehicle for more complex attacks.

Another common problem is represented by unauthorized intrusion in systems. *Intrusion Detection Systems* (IDS) cope with the issue of detecting *unauthorized accesses* to computer systems and computer networks. An *intrusion* can be defined as an attempt by an outsider to gain access to the target system (local or network system). Data Mining methods and algorithms can support the detection phase of known attacks and indeed, from this research trend, a plethora of proposals appeared in recent years. Despite this, classical *sequential algorithms* are not suitable to capture in real time new trends and changes in *streaming data*, which may denote a network intrusion, as they assume that data are static and not changing due to external modifications. In this specific application scenario, *ensemble-based algorithms* (e.g., [62]) supply some specific characteristics of great interest in the context of intrusion detection, and, as a consequence, several *ensemble-based intrusion detection techniques* have been proposed recently. Among well-recognized characteristics of such class of Data Mining algorithms, some that make them particularly suitable to support intrusion detection are the following ones: (i) they can be easily implemented in *parallel/distributed architectures*; (ii) they can improve the accuracy of a *weak learner*; (iii) they can be specialized for the detection of a *particular class*; (iv) they are particularly suitable to the special case of *unbalanced datasets*.

Such type of applications can be implemented effectively with a new paradigm based on combination of multiple models.

The ensemble [17, 44] is a learning paradigm where multiple component learners are trained for the same task by a learning algorithm, and the predictions of the component learners are combined for dealing with new unseen instances. Among the advantages in using ensemble of classifiers, they help to reduce the variance of the error, the bias, and the dependence from a single dataset; furthermore, they can be build in an incremental way and they are apt to distributed implementations. They are also particularly suitable for distributed intrusion detection, because they permit to build a network profile by combining different classifiers that together provide complementary information. However, the phase of building of the ensemble could be computationally expensive as when new data arrives, it is necessary to restart the training phase.

To this aim, this thesis proposes a more flexible approach, and designs a distributed Genetic Programming (GP) framework, based on the distributed Cellular Genetic programming (CAGE) environment [36], to evolve a function for combining the classifiers composing the ensemble, having some attractive characteristics. First, the models composing the ensemble can be trained only on a portion of the training set, and then they can be combined and used without any extra phase of training. Moreover the models can be specialized for a single class and they can be designed to handle the difficult problems of unbalanced classes and missing data.

Furthermore, in case of changes in the data, the function can be recomputed in an incrementally way, with a moderate computational effort and, in a streaming environment, drift strategies can be used to update the models.

In addition, all the phases of the algorithm are distributed and can exploits the advantages of running on parallel/distributed architectures to cope with real time constraints.

This framework is oriented and specialized towards cyber security applications. For this reason, the algorithm is designed to work with different type of input such as the missing data and unbalanced classes, and with various models like one specialized on some tasks or models working with streaming data. All the approaches described above are parts of the general framework, while two specializations, to be applied in the previously illustrated scenarios, are presented. The first is an architecture for a system to detect user profiles. The second one is an new software architecture of a distributed intrusion detection system.

## 1.1 Thesis Overview

The Chapter 2 of the thesis describes some cyber security applications that motivated this research. In particular, a system to identify user behavior and an intrusion detection system are presented as two application scenarios that can be implemented using the proposed framework.

Then some information about the techniques used to build the architecture and the model are provided in the Chapter 3 along with an introduction to the problem of missing data.

The Chapter 4 shows the most correlated works in scientific literature providing the notions behind the formulation of the algorithm to build the ensemble; in addition, it shows the state-of-the-art of the distributed IDS. Moreover, it presents an overview of the main architectures to build a network intrusion system along with an overview of the main ensemble-based techniques implemented in such type of systems.

The Chapter 5 presents the architecture of a framework for combining an ensemble of classifiers. First of all, a definition of the problem and the description of the ensemble approach is presented along with the definition of the components of the system: the base classifiers, the combination function, the tool to build the combination function and the working principles of the approach. Then, the details of the application scenarios addressed by this approach are described: in particular, an optimization to handle missing data and unbalanced classes are the first improvement of the base algorithm. The remaining part of the Chapter presents a solution for handling streaming data and some specialized models useful to detect different classes of attacks. Lastly, a software architecture for a distributed intrusion detection system is described.

The Chapter 6 shows the framework experimental results on artificial and some real datasets proving the effectiveness of the proposed approach.

Finally, Chapter 7 reviews the contributions of this work, and outlines future research to be conducted.

## 1.2 Selected and relevant publications

The relevant publications inspired by this research are listed below.

- Folino, G., Pisani, F., Trunfio, P. (2014). Efficient discovery of data mining services over DHT-based overlays. 2014 International Conference on High Performance Computing & Simulation (HPCS), IEEE Computer Society, Bologna, Italy, pp. 484-490, July 2014. doi: 10.1109/hpcsim.2014.6903724
- Folino, G., Pisani, F. S. (2014). Automatic offloading of mobile applications into the cloud by means of genetic programming. *Applied Soft Computing*, 25, pp. 253–265. doi: 10.1016/j.asoc.2014.09.016
- Folino, G., Pisani, F. S. (2014). Modeling the Offloading of Different Types of Mobile Applications by Using Evolutionary Algorithms. *EvoApplications 2014*, Lecture Notes in Computer Science, Springer, Granada, Spain, pp. 86-97, April 3-5, 2014, 86–97. doi: 10.1007/978-3-662-45523-4\_8
- Folino G, Pisani FS (2015). Combining Ensemble of Classifiers by Using Genetic Programming for Cyber Security Applications. In: *EvoApplications 2015*. Copenhagen, Denmark, April 2015, doi: 10.1007/978-3-319-16549-3\_5
- Folino G, Pisani FS (2016). Evolving meta-ensemble of classifiers for handling incomplete and unbalanced datasets in the cyber security domain. *Applied Soft Computing*, vol. 47, p. 179-190, ISSN: 1568-4946, doi: 10.1016/j.asoc.2016.05.044
- Folino G, Pisani FS, Sabatino P (2016). A Distributed Intrusion Detection Framework Based on Evolved Specialized Ensembles of Classifiers. In: *EvoApplications 2016*. p. 315-331, ISBN: 978-3-319-31203-3, Porto, Portugal, April 2016, doi: 10.1007/978-3-319-31204-0\_21
- Folino G, Pisani FS, Sabatino P (2016). An incremental ensemble evolved by using genetic programming to efficiently detect drifts in cyber security datasets. In: *GECCO (Companion)*. p. 1103-1110, Denver, Colorado, July 20-24, 2016, doi: 10.1145/2908961.2931682
- Guarascio M, Pisani FS, Ritacco E, Sabatino P (2016). Profiling Human Behavior through Multidimensional Latent Factor Modeling. *New Frontiers in Mining Complex Patterns - 5th International Workshop, NFMCP 2016*, Lecture Notes in Computer Science, Springer 2016 (to appear).
- Folino G, Pisani FS (2017). A Software Architecture for Classifying Users in e-Payment Systems. In: *ITASEC*, Venice, Italy, January 17-20, 2017.



## Cyber security applications

Two main common questions, in the cyber security field, concern “how to protect a system” and “how to protect the users of a system”. The first sentence is referred to technologies and policies to protect a system. The second one is related to protection of the users and to the prevention of malicious activities mainly caused by account violations. Obviously, the protection against external and malicious intrusion is the main priority of all sys admin, today, many security breaches are caused by escalation privileges from normal user account. Moreover, the damages caused from identity/credential thefts are growing thanks to social engineering techniques and phishing attacks.

Many types of systems in the industry, in communication and in cyber-physical systems have security issues to handle.

Frequently, the word cyber security is related with credit card fraud and identity theft. In the case of credit card fraud, an attacker obtains a person’s credit card and uses it to make unauthorized purchases. An illicite use of a system can be observed also with telephone calling cards. In this type of attack, someone has access to the physical sim-card or he clones the phone number to impersonate a target person with important security implication.

A more serious risk is the identity theft. In this case, someone assumes the identity of another person by acquiring key personal information such as social security number, and uses that information to carry out transactions under the other person’s name. Even a single such transaction, such as selling a house and depositing the income in a fraudulent bank account, can have devastating consequences for the victim. By the time the owner finds out it will be far too late.

One example of critical physical system that have security implications is the smart grid, a new form of electricity network with high fidelity power-flow control, self-healing, and energy reliability. The process to conduct a risk analysis and to upgrade the procedure to assure an high level of security is very complicated. It requires significant dependence on intelligent and secure communication infrastructures. It requires security frameworks for distributed communications, pervasive computing and sensing technologies in smart grid. The cost of an attack is very high, indeed it

could lead to unreliable system operations, causing unnecessary expenditure, even consequential disaster to both utilities and consumers.

Many of these problems can use data mining techniques to deal with cyber security issues. For instance, considering the attacks to building and critical infrastructures such as power grids and telecommunication systems, these techniques could be applied to identify suspicious individuals and groups, and to discover which individuals and groups are capable of carrying out malicious activities. They can be used also for protecting computer and network systems from corruption due to malicious software or in softwares for intrusion detection and auditing.

To this aim, in this Chapter two scenarios are presented as use cases for cyber security applications. In both cases, a distributed implementation in a platform ensemble-based can achieve excellent results and guarantee a boost of performance with a low resources requirement. The first example shows a system to divide users in groups with same behavior and same weakness. The second example is a distributed architecture for an intrusion detection system.

## 2.1 Users profile classification

Many works present in literature [88], [99] remarked that most serious threats and vulnerabilities concern the user and its wrong behaviors. In modern payment systems, the user is often the weakest link in the security chain. Indeed, in 2015, according to Kaspersky Lab research<sup>1</sup>, 73% of organizations had internal data breach issues that can severely compromise their business. Typical examples are the use of insecure passwords, saving critical data not encrypted on a notebook. In particular for e-payment systems, the users, the operators and even the top managers of these systems are often unaware of the risks associated with vulnerabilities enabled by their behavior and that can be dangerous. Therefore, a system for the protection of payment systems cannot be separated from analyzing the vulnerabilities related to users and the appropriate countermeasures must be undertaken.

In light of these considerations, it is a very hard task to identify the key vulnerabilities associated with the user behavior and to implement a number of measures useful to protect the payment systems against these kinds of vulnerabilities.

A scalable solution for operating with the security weaknesses derived from the human factor, must consider a number of critical aspects, such as profiling users for better and more focused actions, analyze large logs in real-time and also works efficiently in the case of missing data.

Data mining techniques could be used to fight efficiently, to alleviate the effect or to prevent the action of cybercriminals, especially in the presence of large datasets. In particular, classification is used efficiently for many cyber security applications, i.e. classification of the user behavior, risk and attack analysis, intrusion detection systems, etc. However, in this particular domain, datasets often have different number of features and each attribute could have different importance and cost. Furthermore,

---

<sup>1</sup> <https://blog.kaspersky.it>

the entire system must also work if some features are missing. Therefore, a single classification algorithm performing well for all the datasets would be really unlikely, especially in the presence of changes and with constraints of real time and scalability. In the ensemble learning paradigm [17][44], multiple classification models are trained by a predictive algorithm, and then their predictions are combined to classify new tuples.

In order to protect systems, in particular e-payment systems which are the main target for cyber-criminal, from the problems concerning the user behavior, as a proof of concept, a general architecture was designed and it is based on the paradigm of the ensemble. It is useful to monitor the user behavior, divide the users of a payment system into pre-defined classes according to the type of vulnerability enabled and to make possible to address suitable actions (information campaigns, alerts, etc.) towards targeted users of a specific group. Typically, the data useful to classify the user presents many missing features. To overcome this issue, a classification tool was also used, based on artificial intelligence and adopting a meta-ensemble model to operate efficiently with missing data.

### 2.1.1 Methods for classifying user profiles

The inspiration of the approach to design an architecture for a system to assign a profile of risk to each user of a service comes from a project on cyber security, in which one of the main tasks consists in dividing the users of an e-payments systems into homogenous groups on the basis of their weakness or vulnerabilities from the cyber security point of view. In this way, the provider of an e-payment system can conduct a different information and prevention campaign for each class of users, with obvious advantages in terms of time and cost savings. In addition, specialized security policies can be conducted towards the users of a specific class.

This technique is usually named segmentation, i.e. the process of classifying customers into homogenous groups (segments), so that each group of customers shares enough characteristics in common to make it viable for a company to design specific offerings or products for it. It is based on a preliminary investigation in order to individuate the variables (segmentation variables) necessary to distinguish one class of customers from others. Typically, the goal is to increase the purchases and/or to improve customer satisfaction.

Different techniques can be employed to perform this task; in order to cope with large datasets, the most used are based on data mining approaches, mainly clustering and classification; anyway, many other techniques can be employed (see [48] for a survey of these techniques). The authors describe the result of the traditional machine learning techniques applied to a collection of data representing the users interests. The main downside of that approach is the limited reuse of the model learned. The WebMate algorithm is an incremental clustering approach adopted to understand topic of a set of web documents. The WebAce algorithm maps visited URL of a group of users to a vector and cluster together the users with "similar" behavior: i.e. users visiting same sites. Both approaches are limited to identify user profile in term of users visiting pages with shared topic or users visiting same sites. The models

are not easily adaptable to new contexts. Same consideration can be applied to [54] in which the algorithm requires a sequence of predefined actions with an associated frequency.

Another issue to be considered in order to construct the different profiles is the information collection process used to gather raw information about the user, which can be conducted through direct user intervention, or implicitly, through software that monitors user activity. Finally, profiles maintaining the same information over time are considered static, in contrast to dynamic profiles that can be modified or improved over time [46].

In the general case of computer user profiling, the entire audit source can include information from a variety of sources, such as command line calls issued by users, system calls monitoring for unusual application use/events, database/file accesses, and the organization policy management rules and compliance logs. The type of analysis used is primarily the modeling of statistical features, such as the frequency of events, the duration of events, the co-occurrence of multiple events combined through logical operators, and the sequence or transition of events. An interesting approach to computer user modeling is the process of learning about ordinary computer users by observing the way they use the computer. In this case, a computer user behavior is represented as the sequence of commands she/he types during her/his work. This sequence is transformed into a distribution of relevant subsequences of commands in order to find out a profile that defines its behavior. The ABCD (Agent behavior Classification based on Distributions of relevant events) algorithm discussed in [54] is an interesting approach using this technique.

## 2.2 An introduction to intrusion detection systems

This Chapter shows some common applications in the cyber security fields. The intrusion detection systems are an example of that applications and the distributed implementations are very attractive, so one feature of the proposed framework is to detect network intrusions. To this aim, an analysis of the main characteristics of an intrusion detection system together with the most common implementations are presented in the following.

### 2.2.1 The problems of modern NIDS architecture

Network Intrusion Detection Systems (NIDS) are usually distinguished in *signature-based* or *misuse-based* intrusion detection systems and *anomaly-based* intrusion detection systems. In the misuse based approach, that can be summarized as “detect what I know”, when an attack has been discovered, the different steps of the attack are encoded in a *signature*, then the signature is stored in a database and finally it is used by the system to detect the attacks. On the contrary, anomaly-based systems, first try to model the normal behavior of the system to be protected, and then generate an alarm when a significant deviation from this normal behavior is detected. However, it is problematic to define opportunely how significant the deviation is in

these kinds of systems, as it is necessary to find a trade-off between generating a large number of false alarms and letting some attacks elude the system.

As a consequence of our interconnected society and also due to the increased speed of the underlining network connections (gigabits per second) [63], computer network activities, human actions, systems, etc. generate large amounts of data and network traffic (typical examples of these streaming data in this particular domain are network logs, credit card transactional flows, and sensor network data). Indeed, this aspect must be seriously taken into account in designing modern NIDSs, which have to handle these large and fast-changing data. Typically, data mining-based algorithms are among the most used to this aim [33]. However, the ever-changing nature, the high volume of these data, the large amount of storage necessary, etc. can put in crisis traditional sequential data mining algorithms, as they are not able to capture new trends in the stream. In fact, traditional algorithms assume that data is static, i.e. a concept, represented by a set of features, which does not change because of modifications of the external environment.

A more flexible approach is that of ensemble-based algorithms [62], which have been stimulating an increasing interest in the intrusion detection community as they have some characteristics of interest in this particular domain. Indeed, they can be easily implemented in parallel/distributed architectures, improve the accuracy of a weak learner, can be specialized for the detection of one particular class and are particularly suitable to the case of unbalanced datasets.

In the last few years, distributed computing has considerably changed due to the emergence of new environments, such as grid and cloud computing. In particular, the Cloud Computing paradigm has enormous potential for intrusion detection systems. The availability of distributed resources on demand permits data and computational power to be accessible, previously hardly obtainable, and make it possible to deal with complex problems with real-time constraints. Furthermore, even for industries or institutions that do not have the opportunities to use expensive clusters, it is possible to design and to execute massively parallel algorithms also by exploiting the potentialities of the modern multicore CPU and GPU-based architectures.

The increase in the volume of data that need to be processed tends to push the computational load that the underlying hardware needs to sustain to its limits. Indeed, recall that in order to be really useful a NIDS should work in real time, and it needs to sustain the analysis of the data stream involved without saturating all the resources available.

Moreover, increased network connections also mean an increase in the number of alarms generated, especially in the case of signature-based NIDS. As a consequence, there is an ever increasing need for solutions that do not overwhelm human operators with a number of unmanageable alarms. Finally, NIDSs need to be capable of dealing with new undocumented attacks, for which a signature is not already available, i.e., the 0-day attacks.

### **2.2.2 NIDS environment and ensemble-based techniques**

In the field of cyber security applications, the NIDS are one of the main systems that can see an enhancement using ensemble-based algorithm. Detecting malicious action with data mining techniques is a growing trend because they can handle efficiently complex and distributed attack. So, the ensemble-based algorithms combined with modern data mining techniques can be implemented in a parallel/distributed environment with success and in a straightforward way. In this section the main methods for the intrusion detection systems are reviewed to rough out the state-of-the-art.

More in detail, this review aims not only to analyze supervised data mining algorithms, but also works based on anomaly detection and, in particular on clustering suitable to operate in an environment that requires the analysis of data streams in real-time. Sharing knowledge across multiple nodes is another of the key points in designing appropriate NIDS systems and for this reason, collaborative IDS were also included in this study.

## Background

Before the presentation of the framework, some information about the techniques applied in the implementation and some background on the missing data problem are provided in this Chapter. In particular, the technique used to implement the proposed framework is based on Genetic Programming and it is used to compute the combining function of the ensemble. Then an introduction of the ensemble paradigm is also provided.

### 3.1 Evolutionary algorithms and Genetic Programming

Evolutionary algorithms (EA) are heuristics that mimic the processes of natural evolution in order to solve global search problems. Whereas a more traditional optimization technique starts from a single solution with iterative improvements, the EA algorithms search in a "population" of solutions to find an optimum. The search is an application of different operators like recombination, crossover and mutation, each one generate new solutions that are biased towards different regions of the search space. In the following reference is made to a particular branch of EA: the Genetic Programming.

Genetic programming (GP) [61] is an extension of genetic algorithms (GAs) that iteratively evolves a population of (typically) trees of variable size, by applying variation operators. Each individual encodes a candidate solution and is associated with a fitness value that measures the goodness-of-fit of that solution.

The combination function of the ensemble used in the architecture defined in this thesis is a tree of simple operators. The choice to use a GP tool is a natural consequence of the tree nature of the combination function.

GP starts from a population of computer programs and make only best fit programs survive during the evolution. The programmer of the system must choose the functions and the terminals necessary to solve the problem and a fitness function that represents the program's ability to perform the task. The search space will be composed from all the possible programs generated recursively from the functions and the terminals chosen. A computer program (individual) is represented as a parse tree.

Genetic programming uses four main steps to solve problems:

1. Generate an initial population of random compositions of the functions and terminals of the problem (computer programs).
2. Execute each program in the population and assign it a fitness value according to how well it solves the problem.
3. Create a new population of computer programs by applying genetic operators (mutation, crossover, etc.) to some selected tree (best fit trees are selected most likely)
4. The best computer program that appeared in any generation, the best-so-far solution, is designated as the result of genetic programming.

Steps 2 and 3 are repeated until a maximum number of generations is reached or a termination criterium is met (problem is solved exactly or error is less than a threshold).

The most frequent operators of GP are mutation and crossover applied to a tree. The first one chooses a random point in the tree and it create a new random tree linked to the that point. The crossover operator chooses two random points in two different trees and exchanges the respective subtree.

Large sizes of population and a sufficient number of generations are the main problems of the GP. Also the necessity of high computational resources, both in terms of memory, to store large populations of trees, and in terms of time, to evaluate the fitness of the individuals in the population, may degrade the GP performance drastically or make the algorithm inapplicable when it must cope with large difficult problems. The modern distributed architectures partially solve or alleviate this problem, for this reason the GP tools are a good choice to find optimum or sub-optimum solutions.

### 3.1.1 Parallel implementation of a GP tool

The generation of combining function is computed by a tool based on Genetic Programming. A simple option is represented by a distributed tool named Cage. Cage is based on a cellular model to parallelize GP. The cellular model is fully distributed with no need of any global control structure and it is naturally suited for implementation on parallel computers. In this model, the individuals of the population are located on a specific position in a toroidal two-dimensional grid and the selection and mating operations are performed, cell by cell, only among the individual assigned to a cell and its neighbors. This local reproduction has the effect of introducing an intensive communication among the individuals that could influence negatively the performance of the parallel implementation of *GP*. Moreover, unlike genetic algorithms, where the size of individuals is fixed, the genetic programs are individuals of varying sizes and shapes. This requires a large amount of local memory and introduces an unbalanced computational load per grid point. Therefore, an efficient representation of the program trees must be adopted and a load balancing algorithm must be employed to maintain the same computational load among the processing nodes.



*CAGE* implements the cellular *GP* model using a one-dimensional domain decomposition (in the  $x$  direction) of the grid and an explicit message passing to exchange information among the domains. The one-dimensional decomposition permits to achieve a small number of messages sent. The concurrent program that implements the architecture of *CAGE* is composed of a set of identical *slice processes*. No coordinator process is necessary because the computational model is decentralized completely. Each slice process, which contains a portion of elements of the grid, runs on a single *PE* of the parallel machine and updates all the individuals of the subpopulation.

Cage has many parameters that change its behavior such as the population size, the max depth that the trees can have after the crossover, the parsimony factor, the number of iterations, the number of neighbors of each individual, and the replacement policy.

The parsimony factor parameter measures the complexity of the individuals penalizing the solutions with many members. The increase of the value is tied to a more simple solutions, although can affects negatively the results.

The size of the subpopulation of each slice process is calculated by dividing the population for the number of the processors on which *CAGE* is executed. Each slice process updates the individuals belonging to its subgrid sequentially.

Because of the data decomposition, physically neighboring portions of data are allocated to different processes. To improve the performance and to reduce the overhead due to the remote communications, a local copy of boundary data in each process is introduced. This avoids remote communication more than once on the same data.

### 3.2 The problem of missing data

One of the main problems of modern cyber security applications is how to handle the missing data. Different missing patterns can be considered to model the problem. Missing completely at random (MCAR), to describe data, in which the complete cases are a random sample of the originally dataset, i.e., the probability of a feature being missing is independent of the value of that or any other feature in the dataset; missing at random (MAR) describe data that are missing for reasons related to completely observed variables in the data set. Finally, the MNAR case considers the probability that an entry will be missing depends on both observed and unobserved values in the data set. Therefore, even if MCAR is more easy to handle, the most interesting case is the MAR case, as it is a more realistic model and it is suitable to many real-world applications, i.e. weather forecast with multiple source of data and each one can be missing (many features belong to each sources).

Data mining and in particular classification algorithms must handle the problem of missing values on useful features. The presence of missing features complicates the classification process, as the effective prediction may depend heavily on the way missing values are treated. The performance is strictly related to rates of missing

data: a low rate (typically less than 5%) is generally considered manageable, while higher rate can be very problematic to handle.

In a scenario where the data comes from multiple sources, the most common hypothesis is to consider the missing values are present in both the training and the testing data as the same sources of data are not available for all the instances of the dataset. However, without any loss of generality, it is more easy to suppose that the training dataset is complete. Even in the case of the presence of a moderate number of tuples presenting missing data, it can be reported to the previous case, simply by deleting all the incomplete tuples. However, handling missing data by eliminating cases with missing data will bias results, if the remaining cases are not representative of the entire sample. Therefore, different techniques can be used to handle these missing features (see [67] for a detailed list of them).

The easiest way to handle missing data is the listwise deletion method: only complete records are retained, others are removed from dataset. Although it is very simple, it has two main drawbacks. Firstly, with real data where missing are frequently many tuples must be discarded and the worst scenario is a dataset with no complete tuple: in this case, after the method elaboration, the dataset becomes empty. The second problem is the computation of statistics: the result of a statistical operator is biased even with a very low rate of missing data.

An alternative to deletion is the imputation, i.e. the missing value is replaced by an estimation. The most simple way to find a replacement value is to use the most common value in the attribute domain or the mean value. The result is altered significantly because the model gives a greater importance to a mean value which is not the real mean; indeed, with this approach the standard deviation of the attribute is underestimated.

Some improvements are proposed to mitigate these problems. Multiple imputation computes several values for each missing item, while local imputation methods are based on local knowledge to find the estimation. As example, the hot deck imputation finds an estimation of the attribute within all tuples of the same class. These approaches reduce but not eliminate the drawback of the imputation task but the results depend from the missing rate. When the missing data increase, the effectiveness of the method is reduced.

A statistical approach can be used to estimate a missing value. The EM (Expectation Maximization) algorithm is a well-known approach to estimate the parameters of a PDF (probability density function) of an incomplete sample. This approach presents two main problems: the distribution of the samples could be not known (it is a required parameter of EM) and the method is an iterative algorithm cpu-intensive not suitable for large dataset.

The above-mentioned strategies handles missing data by removing any tuple with missing values or replacing the missing value with a meaningful estimate. Another approach is to build models can handle missing data natively. Indeed, with MAR type problems where groups of features are missing (which is the focus of this thesis), multiple classifiers can be trained on dataset partition that have no missing data and the classification is the result of the combination of single predictions. In this way,

the model can handle missing data directly without the need of replacing/estimating the missing value.

To understand the MAR problem in which multiple sources are combined to form the input of the system, the problem is described as following.

Given  $D_1, D_2, \dots, D_k$  datasets; typically each dataset comes from a different source of data, but can be used to predict the same class. Therefore, the corresponding  $i_{th}$  tuple of the different datasets can be used to predict the class of the same instance. However, a particular tuple of a dataset can be missing, i.e., all the features belonging to the same source of data of that tuple are missing.

However, without any loss of generality, even a problem of missing features of an incomplete dataset can be reported to the previous one, by grouping tuples with the same missing features.

If we consider a dataset

$$D = \{(x_{1i}, x_{2i}, \dots, x_{di}), i = 1..N\}$$

the dataset is incomplete if at least one entry in  $[1, d]$  is missing.

For instance, consider the incomplete dataset represented in Table 1 consisting of 6 tuples and 5 features.

**Table 3.1.** An incomplete dataset of 6 tuples and 5 features.

N	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$
1			?	?	
2		?			
3					
4			?	?	
5		?			
6					

This dataset can be partitioned in complete datasets by grouping features having the same missing features. A possible partition could be the following;  $D_1 = \{1, 4\}$  considering the features  $x_1, x_2$  and  $x_5$ ,  $D_2 = \{2, 5\}$  considering the features  $x_1, x_3, x_4$  and  $x_5$  and  $D_3 = \{3, 6\}$  considering all the features. Then, each complete dataset obtained can be used as training set for a classifier algorithm and the different models obtained can be used to classify each tuple. The problem of decomposing the dataset could become complex whether the missing features follows a random pattern and cannot be easily grouped in order to decompose the original datasets in a few complete datasets. In this case, the technique illustrated in this thesis is not adequate. However, it is difficult to find a single technique that can handle all types of missing feature. In this thesis the focus is on a pattern of missing data that covers a large number real applications in which a group of features come from the same source of data and potentially can be all missing.

### 3.3 Ensemble-based techniques

For the classification problems, the ensemble-based techniques permit to combine multiple (heterogeneous or homogeneous) models in order to classify new unseen instances. In practice, after a number of classifiers are built usually using some parts of the original dataset, the predictions of the different classifiers are combined and a common decision is taken. Different schemas can be considered to generate the classifiers and to combine the ensemble, i.e. the same learning algorithm can be trained on different datasets or/and different algorithms can be trained on the same dataset.

The best known and used ensemble-based algorithm is the boosting algorithm, first introduced by Schapire [83] and Freund [84]. In order to boost the performance of any “weak” learning algorithm, i.e. an algorithm that “generates classifiers which need only be a little bit better than random guessing” [84], the boosting method adaptively changes the distribution of the training set according to how difficult each example is to classify. This algorithm, in common with many ensemble-based algorithms, has the drawback of needing to repeat the training phase for a number of rounds and that is really time-consuming in the case of large datasets or not practicable in the case of real-time requirements, i.e. in the intrusion detection domain. On the contrary, other variants of ensemble-based algorithms use functions to combine the classifiers that do not need to reuse the original training set. The majority vote is a classical example of this kind of combiner function, but there are many other functions with this property and they are named non-trainable combiners [62]. Combiner functions having this property considerably reduces the time requirements needed to compute them, as it is not necessary to reuse the training set in this phase.

The information fusion approach, which combines information coming from different sources in order to build new aggregate features or to enhance the solution of a problem, may permit to solve better or more efficiently an IDS task [30]. In fact, different aspects can motivate the use of information fusion for this kind of problems, i.e., information may be present at multiple abstraction levels and may be collected from multiple sources, different abstraction levels can be present in the data, it is effective in increasing timeliness of attack identification and in reducing false alarm rates and usually obtain high detection rate, etc. A typical example of application based on information fusion is a software for sensor data analysis: the data collected by sensors can be generated in different environment and with different features. However, as an additional computational overhead must be also taken into account to process these different sources, a fast and effective fusion of information is fundamental to enhance the value of the classification and clustering task and the computational cost of the process must be taken into account. Many fusion-based techniques can be applied to the intrusion detection task. An interesting classification [8] considers three cases depending whether the algorithm operates at the level of the data (data level), of the features (feature level), and of the final decision (decision level). Indeed, some methods try to consider the semantic groups in which the high dimensionality space of the features is divided or the hierarchical abstraction levels or the type of information contained. Furthermore, as the intrusion detection

task can be computationally expensive, many of the information fusion-based approaches adopt non-trainable functions to combine the classifiers.

An approach of network intrusion detection based on five decision fusion algorithm is presented in [47]. The algorithm combines data of the transmitted content, the information about connections and the statistics about traffic related to the same service. For each source, multiple classifiers are trained and their results are combined by a fusion rule. In the paper, five different method are presented along with an experimental comparison. The reported results showed that dynamic classifier selection approach provides a better trade-off between generalization abilities and false alarm generation.

Another example of algorithm that process multiple sources of data is dLEARNIN [75]. It is an ensemble of classifiers based on a combining function that try to minimize the cost of classification errors. In order to minimize the cost of classification, the algorithm computes a cost matrix on validation data and uses these informations to find the best thresholds for each class. Then the classification of an instances is based only on the classifiers that exceed the threshold; if none or multiple outputs satisfy this requirement, then a voting algorithm is used to find the final decision.

HMMPayl [4] is an example of fusion-based IDS, where the payload is represented as a sequence of bytes; but to reduce the computational complexity, only a small percentage of the sequences (randomly sampled) are used. The analysis is performed using Hidden Markov Models (HMM). HMMPayl follows the Multiple Classifiers System paradigm to provide better classification accuracy, to increase the difficulty of evading the IDS, and to mitigate the weaknesses due to a non-optimal choice of HMM parameters.

IDSs benefit from using information fusion and ensemble-based algorithms for a number of reasons. First, these techniques perform well both when data are very scarce and when we have a huge amount of data; furthermore, they can be easily implemented in efficient computing environments such as parallel, multi-core, and GPGPU architectures and also on P2P and Cloud computing environments. In addition, they can easily model different abstractions or parts of a network, i.e., some models can be trained on some parts or on some levels of the network and finally combined together, to assure a better prediction.

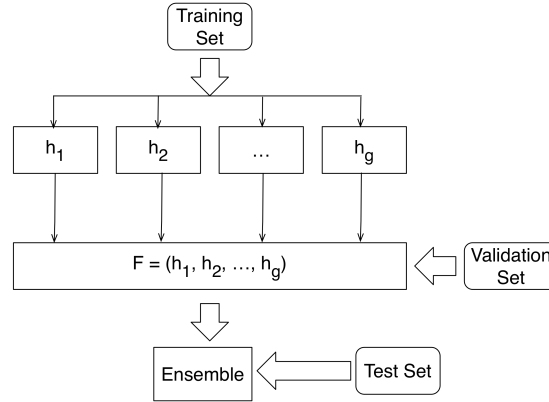
### 3.4 Combining ensemble of classifiers

In this section, a general schema for combining an ensemble of classifiers is shown and the introduction of the concept of "non-trainable functions", that can be used in order to combine an ensemble of classifiers without the need of a further phase of training, is presented. Then, the distributed GP framework used to evolve the combining function of the ensemble is illustrated.

#### 3.4.1 Ensemble of classifiers and non-trainable functions

Ensemble permits to combine multiple (heterogenous or homogenous) models in order to classify new unseen instances. In practice, after a number of classifiers are

built usually using part of the dataset, the predictions of the different classifiers are combined and a common decision is taken. Different schemas can be considered to generate the classifiers and to combine the ensemble, i.e. the same learning algorithm can be trained on different datasets or/and different algorithms can be trained on the same dataset. The framework is based on schema shown in Figure 3.1, in which different algorithms are used on the same dataset in order to build the different classifiers/models.



**Fig. 3.1.** A general schema for combining ensemble of classifiers

A formalization of this approach is described in the following.

Let  $S = \{(x_i, y_i) | i = 1, \dots, N\}$  be a training set where  $x_i$ , called example or tuple or instance, is an attribute vector with  $m$  attributes and  $y_i$  is the class label associated with  $x_i$ . A predictor (classifier), given a new example, has the task to predict the class label for it.

Ensemble techniques build  $g$  predictors, each on a different training set, then combine them together to classify the test set. As an alternative, the  $g$  predictors could be built using different algorithms on the same/different training set.

The combining function is composed by operators that does not need to be trained on data and consequently, the ensemble is ready for operation as soon as the base classifiers are trained. These functions are named non-trainable combiners [62] and could be used as functions in a genetic programming tree.

To understand how the algorithm works, some definitions are useful.

Let  $x \in R^N$  be a feature vector and  $\Omega = \{\omega_1, \omega_2, \dots, \omega_c\}$  be the set of the possible class labels. Each classifier  $h_i$  in the ensemble outputs  $c$  degrees of support, i.e., for each class, it will give the probability that the tuple belong to that class. Without loss of generality, all the  $c$  degrees are in the interval  $[0, 1]$ , that is,  $h_i : R^N \rightarrow [0, 1]^c$ . Denote by  $H_{i,j}(x)$  the support that classifier  $h_i$  gives to the hypothesis that  $x$  comes from class  $\omega_j$ . The larger the support, the more likely the class label  $\omega_j$ . A non-trainable combiner calculates the support for a class combining the support values

of all the classifiers. For each tuple  $x$  of the training set, and considering  $g$  classifiers and  $c$  classes, a Decision Profile matrix DP can be build as follow:

$$DP(x) = \begin{bmatrix} H_{1,1}(x) & \dots & H_{1,j}(x) & \dots & H_{1,c}(x) \\ H_{i,1}(x) & \dots & H_{i,j}(x) & \dots & H_{i,c}(x) \\ H_{g,1}(x) & \dots & H_{g,j}(x) & \dots & H_{g,c}(x) \end{bmatrix}$$

where the element  $H_{i,j}(x)$  is the support for  $j$ -th class of  $i$ -th classifier.

The functions used in this approach simply combine the values of a single column to compute the support for  $j$ -th class and can be defined as follow:

$$\mu_j(x) = F[H_{1,j}(x), H_{2,j}(x), \dots, H_{g,j}(x)]$$

For instance, the most simple function we can consider is the average, which can be computed as:  $\mu_j(x) = \frac{1}{g} \sum_{i=1}^g H_{i,j}(x)$

The class label of  $x$  is the class with maximum support  $\mu$ .





## Related works

In this section, an overview of the related work to evolutionary algorithms and intrusion detection approaches is presented. The study of these techniques is a preliminary work to design of an architecture for the systems proposed in this thesis.

In the first part of this Chapter two topics are addressed. The first topic shows related approaches about missing data and incomplete data. The main works that handle these problems are described. Then, the works about evolutionary algorithms applied to ensemble based model are presented. The same technique is used to combine base classifiers in the proposed framework.

In the second part of the Chapter, related works about ensemble paradigm are presented. Similar approaches as well as the different techniques to combine ensemble are described.

Moreover, the main implementations of distributed IDS are presented and they become the base reference for the design of the IDS architecture presented in Chapter 5. The paragraphs describes different types of IDS: distributed and collaborative implementations, applications based on data mining approaches, and solutions designed for high performance computing.

### 4.1 Works on incomplete datasets and related approaches

In this section, an analysis of the main methods to cope with missing data and incomplete datasets and a general schema for combining an ensemble of classifiers and the concept of "non-trainable functions" that can be used in order to combine an ensemble of classifiers without the need of a further phase of training is presented.

#### 4.1.1 Incomplete datasets and missing data

Several approaches use the ensemble as method to cope with incomplete and/or unbalanced data. Most of the analyzed approaches employ a high number of resources to generate the function and therefore, their usage is not particularly recommended

for large real datasets. Moreover, most of them require to process the training set in the phase of generating the combiner function, with a considerable overhead. Other approaches do not use the training set, however, they build a number of random subsets of the features of the original dataset and therefore, its application is problematic when the number of features is high. In this Section some examples of that approaches are introduced and an analysis of the missing data problem is presented.

Chen et al. [24] use multiple ensembles to classify incomplete datasets. Their strategy consists in partitioning the incomplete datasets in multiple complete sets and in training the different classifiers on each sample. Then, the predictions of all the classifiers could be combined according to the ratio between the number of features in this subsample and the total features of the original dataset. This approach is orthogonal to our and therefore, it could be included in our system.

Another approach to cope with incomplete datasets can be found in [98]. The authors build all the possible LCP (Local Complete Pattern), i.e., a partition of the original datasets into complete datasets, without any missing features; a different classifier is built on each LCP, and then they are combined to predict the class label, basing on a voting matrix. If a dataset presents missing values for many attributes, this approach requires to build partitions for each combination of attributes: the result is an increasing of the complexity of the overall process.

An LCP is a subset of feature such that a portion of original data expressed in term of that features does not contains missing data. As example, if a dataset is defined by attributes  $\{x_1, x_2, x_3, y\}$  where  $y$  is the label and contains the tuples  $[a(5, 6, ?, 1), b(3, 4, 9, 1), c(?, 3, 5, 0), d(?, 7, 5, 0)]$  where  $?$  denotes missing value; one LCP could be the subset defined by the features  $\{x_1, x_2\}$  and contains the tuple  $[a, b]$ ; another one could be defined by features  $\{x_2, x_3\}$  and contains the tuples  $[b, c, d]$ .

The experiments compared the proposed approach with two techniques to cope with missing data, i.e., deletion and imputation, on small datasets and show how the approach outperforms the other two techniques. However, the phase of building the LCP could be really expensive. Also considering the size and the dimensionality of data collected by modern software, the time and resources required to find and process all possible LCP represent an unacceptable limit.

Learn++.MF [80] is an ensemble-based algorithm with base classifiers trained on a random subset of the features of the original dataset. The approach generates a large number of classifiers, each trained on a different feature subset. Each classifier is trained on instances with no missing data for the selected features subset. In practice, the instances with missing attributes are classified by the models generated on the subsets of the remaining features. The algorithm uses a majority voting strategy in order to assign the correct class under the condition that at least one classifier must cover the instance. When the number of attributes is high, it is unfeasible to build classifiers with all the possible sets of features; therefore, the subset of the features is iteratively updated to favor the selection of those features that were previously undersampled. However, even with this optimization that reduce the number of classifier to train, the number of learners required to achieve good results remains high, particularly when there are many features with missing values, and this limits the real applicability of the approach to datasets with a low number of attributes.

Indeed, the results presented by the authors show the decreasing of the performance when a dataset with many features is processed. These aspect must be addressed by all modern frameworks.

## 4.2 Ensemble and evolutionary algorithms for classification

Evolutionary algorithms have been used mainly to evolve and select the base classifiers composing the ensemble [74, 37] or adopting some time-expensive algorithms to combine the ensemble [87]; however a limited number of papers concerns the evolution of the combining function of the ensemble by using GP, which are illustrated in the following.

Chawla et al. [89] propose an evolutionary algorithm to combine the ensemble, based on a weighted linear combination of classifiers predictions, using many well-known data mining algorithm as base classifiers, i.e. J48, NBTree, JRip, etc. In [23], the authors extend their work in order to cope with unbalanced datasets. In practice, they increase the total number of base classifiers and adopt an oversampling technique. In [90], the authors consider also the case of homogenous ensemble and show the impact of a cut-off level on the total number of classifiers used in the generated model. The approach proposed in Chapter 5 also uses heterogeneous classifiers, but it combines functions of different types, also considering weights derived by the performance of the classifiers on the training set; it takes also into account the effect of unbalanced datasets and, in addition, the method is apt to operate with incomplete datasets, without using oversampling techniques.

Yan Wang et al. [98] use multiple ensembles to classify incomplete datasets. Their strategy consists in partitioning the incomplete datasets in multiple complete sets and to train the different classifier on each sample. Then, the predictions of all the classifiers could be combined according to the ratio between the number of features in this subsample and the total features of the original dataset.

In [1], the authors develop a GP-based framework to evolve the fusion function of the ensemble both for heterogenous and homogeneous ensemble. The approach is compared with other ensemble-based algorithms and the generalization properties of the approach are analyzed together with the frequency and the type of the classifiers presents in the solutions. The main aim of the paper is to improve the accuracy of the generated ensemble, while distributed implementations and the problems concerning incomplete and unbalanced datasets are not explored. In addition, the authors do not consider weights depending from the performance of the classifiers on the datasets.

In [16], Brameier and Banzhaf use linear genetic programming to evolve teams of ensemble. A team consists of a predefined number of heterogeneous classifiers. The aim of genetic algorithm is to find the best team, i.e. the team having the best accuracy on the given datasets. The prediction of the team is the combination of individual predictions and it is based on the average or the majority voting strategy, also considering predefined weights. The errors of the individual members of the team are incorporated into the fitness function, so that the evolution process can find the team with the best combination of classifiers. The recombination of the team members is

not completely free, but only a maximum pre-defined percentage of the models can be changed. In order to have a more flexible system, a GP module should generate tree-based models and the number of base classifiers in the tree should not be pre-defined; therefore, it will be the evolution process to select the best combination of the base classifiers.

### 4.3 Ensemble technique for cyber security applications

Ensemble [17, 44, 62] is a learning paradigm where multiple component learners are trained for the same task by a learning algorithm, and the predictions of the component learners are combined for dealing with new unseen instances. Among the advantages in using ensemble of classifiers, they help to reduce the variance of the error, the bias, and the dependence from a single dataset; furthermore, they can be build in an incremental way and they are apt to distributed implementations. Finally, for the particular task of intrusion detection, they are able to combine different classifiers that together provide complementary information and also are particularly apt to handle unbalanced classes.

In [60], specialized detectors were successful employed to improve the classification accuracy of malware detection. The detectors are combined in an ensemble using standard pre-defined functions, i.e., majority voting, stacking, etc. Therefore, the approach is not easily adaptable to the case of data stream and when the data change. The adoption of non-trainable functions for cyber security problems in general, and also for the intrusion detection, is also explored in [35], in which, the non-trainable functions were used for handling missing data.

In [1], the authors develop a GP-based framework to evolve the fusion function of the ensemble both for heterogenous and homogeneous ensemble. The main aim of the paper is to improve the accuracy of the generated ensemble, while distributed implementations and the problems concerning incomplete and unbalanced datasets are not explored. The authors do not consider weights depending from the performance of the classifiers on the datasets.

#### 4.3.1 Data stream and time window

A stream of data is a particular data source that can be seen as an infinite data sets. With this model, it is not possible to have a "global" view of the stream. Hence, a partition method is required. Usually, a fragment or a chunk of data are collected and analyzed. The size of that chunk is a parameter and it can be measured by the number of instance, by a time interval (all data observed in 5 minutes, hours, etc.) or by a quantity that depends from the domain/use case. In the literature, a chunk of data is named window. Independently of the method used to build a window, many algorithms are processed window-by-window: i.e. they are trained on a window and updated on the next one. The most common types of window are:

- fixed window, in which the size of instances in each window is fixed;

- sliding window, in which the size is fixed but the content depends from a time interval, after each step oldest data are removed and newest data are added;
- windows based on sessions, in which the data comes from a session defined by the nature of the data and each one has an identifier.

A simple use of a window can be found in [29]. In the paper the ensemble classifier is constructed from the one-class base classifiers for mining data streams with a concept-drift. The stream is processed in chunks and the instances are selected applying the one-class k Nearest Neighbor method. The classification process is based on a weighted combination of the base classifiers. The classifier's weight is calculated from its accuracy and the length of time spent as a member of the ensemble. Many classifiers are trained on each chunks and the classifiers which have a weight above the mean values along with the ones used in the last update are maintained in the ensemble.

In the [79] the authors propose a method to add/remove classifiers to/from ensemble depending on results on a chunk of the stream data. The decision is based of an improvement of the accuracy when the classifier is added to ensemble and on the absence of effects (in term of decreasing of accuracy) for the removing case. The method calculates a probability as an approximation of a binomial distribution proving that the accuracy obtained on a single chunk is valid for the stream.

#### 4.3.2 Ensemble technique and IDS systems

In the cyber security domain, computer and network technologies have intrinsic security weaknesses, i.e., protocol, operating system weaknesses, etc. In addition, computer network activities, human actions, etc. generate large amounts of data, hard to handle without using ad-hoc designed algorithms and distributed machines.

*Intrusion Detection Systems (IDS)* cope with the issue of detecting *unauthorized accesses* to computer systems and computer networks. An *intrusion* can be defined as an attempt by an outsider to gain access to the target system (local or network system). Data Mining methods and algorithms can support the detection phase of known attacks and indeed, from this research trend, a plethora of proposals appeared in recent years. Despite this, classical *sequential algorithms* are not suitable to capture in real time new trends and changes in *streaming data*, which may denote a network intrusion, as they assume that data are static and not changing due to external modifications. In this specific application scenario, *ensemble-based algorithms* (e.g., [62]) supply some specific characteristics of great interest in the context of intrusion detection, and, as a consequence, several *ensemble-based intrusion detection techniques* have been proposed recently [39]. Among well-recognized characteristics of such class of Data Mining algorithms, some that make them particularly suitable to support intrusion detection are the following ones: (i) they can be easily implemented in *parallel/distributed architectures*; (ii) they can improve the accuracy of a *weak learner*; (iii) they can be specialized for the detection of a *particular class*; (iv) they are particularly suitable to the special case of *unbalanced datasets*.

Moreover, building the ensemble could be computationally expensive. For this reason, a wide area of research is dedicated to study of methods for executing efficiently this task. Evolutionary algorithms are an interesting solutions. They have been used mainly to evolve and select the base classifiers composing an ensemble [74, 37] or adopting some time-expensive algorithms to combine an ensemble [87]. Chawla et al. [89] propose an evolutionary algorithm to combine the ensemble, based on a weighted linear combination of classifiers predictions, using many well-known data mining algorithm as base classifiers, i.e. J48, NBTree, JRip, etc. Yan Wang et al. [98] use multiple ensembles to classify incomplete datasets. Their strategy consists in partitioning the incomplete datasets in multiple complete sets and to train the different classifier on each sample. Then, the predictions of all the classifiers could be combined according to the ratio between the number of features in this subsample and the total features of the original dataset. In [1], the authors develop a GP-based framework to evolve the fusion function of the ensemble both for heterogeneous and homogeneous ensemble. The approach is compared with other ensemble-based algorithms and the generalization properties of the approach are analyzed together with the frequency and the type of the classifiers presents in the solutions.

Meanwhile previous works are mainly dedicated to solve the general problem of computing the combining function with genetic programming, some interesting solutions are more oriented to design of a network anomaly detection systems, for instance [81] and [58]. Moreover, an interesting work in the field of anomaly-based network intrusion detection can be found in [45], which also includes a panoramic of the principal anomaly-based intrusion detection techniques, namely statistical-based, machine learning-based and knowledge-based. For each of the techniques illustrated, the main systems present in the literature are described together with the main challenges of the field.

However, one of the most comprehensive introductions to the field of *Network Anomaly Detection* (NAD) is presented in [8]. The main aim of this paper is to provide enough informations for a new researcher to acquire a certain familiarity with every aspect of the field. Accordingly, it contains a clear categorization of most common attacks encountered and of the main systems for network intrusion detection, according to the anomaly detection method and to the computational technique adopted. Based on this classification, an in-depth comparison is performed of the different architectures. A review of the principal tools that are useful to researchers in the network anomaly detection field is also included, as well as a comparative description of the datasets of network traffic publicly available.

Other works, described in the following, are more specific to a topic, types of data or technique. The problem of preprocessing and feature selection in the anomaly detection field is illustrated in detail in [31]. The review covers both automated methods for features extraction and techniques used to reduce the dimensionality of the problem. In fact, the authors analyze the literature on this topic and identify a shift from techniques that rely on the analysis performed by human domain experts to automatic solutions that implement a broad range of techniques of analysis.

Generally, systems are categorized according to the detection paradigm considered and to the type of attack, i.e., denial of service, worms, etc. The work in [86]

concentrates on the intrusion detection systems that operate at the level of network flow. In [76], the authors present a review of the main machine learning techniques employed in the network intrusion detection field. The techniques considered include rule-based learning, decision tree, Bayesian reasoning, neural networks, support vector machines and clustering. Moreover, a section devoted to nature-inspired techniques is also included, which covers artificial immune systems, genetic programming and swarm intelligence. In addition, a comparison of the performance of the different techniques is illustrated, reporting experimental results contained in the literature. Finally, the current challenges of the field are discussed.

In [50], a panoramic of tools useful to researchers in the intrusion detection field is presented. In particular, the main steps necessary to perform an attack are described. Scanning and sniffing tools are discussed, as well as an extensive discussion of attack launching tools. Network monitoring systems and working attack detection systems are also covered. Finally, the main challenges that researchers in the field have to face are the subject of the closing considerations of the work. The same authors also reviewed the topics of port scans, scanning tools and launching attacks in [9] and the main detection techniques in [11, 10].

An in-depth analysis of network intrusion detection systems from the distributed point of view can be found in [103] and [93]. Zhou et al. [103] point their attention to coordinated attacks such as large-scale scans, worm diffusion and distributed denial of service and on the research in the field of collaborative intrusion detection that aims to detect and prevent these kinds of attack. The work describes the different architectures of a collaborative intrusion detection system and also illustrates some algorithms used for the task of alert correlation. Vasilomanolakis et al., [93] outline the general architecture of a collaborative intrusion detection system. The main building blocks of these systems and the methodology to integrate them in a common environment are described. In comparison with the latter two papers, the work proposed in this thesis is more focused on the ensemble-based paradigm and supervised data mining techniques. In addition, the analysis is focused on the main distributed algorithms used for these topics, while the above-cited papers are more aimed at a general discussion of the possible collaborative and distributed architectures, without discussing the interesting ensemble-based techniques.

Another related work [25] reviews many state-of-the-art solutions for computer security from the point of view of the information fusion and proposes a general scheme to perform information fusion for intrusion detection in computer systems. In addition, the work discusses strengths and weaknesses of currently used approaches and examines some interesting research issues. However, the paper treats computer security in general, and furthermore it does not analyze parallel/distributed solutions.

#### **4.4 IDS techniques in distributed environment**

In this Section, the main techniques and some IDS with a distributed implementation are presented. The different types of IDS include collaborative IDS, IDS based on data mining algorithms and IDS implementation for high performance computing.

#### 4.4.1 Distributed and collaborative IDS

Attacks conducted against online systems which occur in multiple networks simultaneously (i.e. large-scale stealthy scans, worm outbreaks and distributed DDoS), are among the most dangerous. In addition, coordinated attacks are really difficult to detect using isolated intrusion detection systems since these systems usually monitor only a small portion of a network. Moreover, it is often the case that a particular IDS is more efficient in detecting certain type of attacks, while a different one is more suitable for other type of attacks. For instance, it is well known that misuse based IDSs are very efficient in detecting known attacks based on a database of signatures, but they are rather weak in detecting new attacks. On the contrary, anomaly based IDSs are able to detect a certain number of new attacks but they usually show a rather high false positive rate even in the case of known attacks. As already remarked, architectures of IDSs based on the anomaly detection paradigm implement relatively complex analysis techniques in order to detect outliers in the network traffic being monitored. For instance, these techniques can be either data mining based or statistical based, but in any case, in order to attain a suitable detection rate and to minimize the false positive rate, they usually involve a rather high computational complexity. As a consequence, in order to cope with the high volume data involved in a real time environment, the natural choice is to exploit the potentiality of modern parallel hardware. Finally, detecting anomalies in network traffic often requires the use of time-consuming computational techniques, e.g. data mining and statistical analysis, and it is difficult for them to operate in real-time. Therefore, it is necessary to distribute the computational load required using distributed environments, i.e., parallel machines, cloud computing, etc.

Recently, for the reasons described above, the interest in distributed IDS has grown and different methodologies have been proposed. In the following subsections, Collaborative Intrusion Detection Systems (CIDS) are presented. Basically, in the CIDS approach, the aim is to correlate information on attacks coming from different subnetworks. Next, the attention is directed to solutions based on data mining techniques that, although not all of them are set up and tested in a fully parallel environment, nonetheless the solutions and the algorithms employed are particularly suitable for a parallel implementation. Lastly, a Section is dedicated to a brief discussion on solutions whose implementation takes full advantage of modern readily available parallel hardware.

#### 4.4.2 Collaborative IDS

The approach to intrusion detection based on the collaborative paradigm has the potential to design IDSs capable of detecting intrusions that occur across large networks and to correlate alarms coming from different sensors. Moreover, CIDSs reduce computational costs by sharing intrusion detection resources among different networks and they mitigate the problem of false alarms that would be generated by a single IDS operating alone. The main components of a CIDS, in accordance to the work of



[103], are a *detection unit* and a *correlation unit*. The detection unit consists of multiple sensors deployed in a network that generates low level intrusion alerts. These low level intrusion alerts are combined by the correlation unit, then high level reports are generated and finally the real nature of the attacks is confirmed. The design of a CIDS faces then two main challenges, namely *system architecture* and *alert correlation* methodologies. From the point of view of the architecture, a typical implementation is represented by the *Centralized approach*, in which the data are collected by different sources but analyzed and correlated by a single central unit. This approach suffers from various drawbacks; first, as most of the work is done by a single unit, serious problems of reliability are present, since the failure of the central unit causes the CIDS stops from properly working. Moreover, a large computational power is required in order to prevent the degradation of the performance of the CIDS. In the *hierarchical approach*, the correlation units are structured in a pyramidal way, with layers that perform increasing complex analysis on the data acquired by the detection units, but the final analysis is always performed by a central unit, namely the top of the pyramid. This approach frees the central unit from a part of the computational load, with respect to the centralized approach, but suffers from similar analogous problems to the ones mentioned above. Finally, there is a *fully distributed approach* in which the detection and the correlation work is spread among different units. This approach can be set up by a P2P network, for instance, but it is often hard to coordinate a large number of nodes in order to attain a suitable detection accuracy and to guarantee the scalability of the global system. Indeed, the mechanism implemented to share information should be efficient enough to hinder the speed at which menaces spread across a network; moreover, it should remain so as nodes are added to the system. Just to give a glimpse of how hard this problem is, in some extreme cases, as in the case of the SQL-slammer worm, it has been estimated that as the worm began spreading through the Internet it infected over 90% of the vulnerable hosts just in ten minutes. In the following, the main collaborative IDS and a summary of their characteristics is reported in Table 4.1.

In [102], the authors focus their attention on the problem of alert correlation, with the purpose of improving the scalability without renouncing the accuracy of the system, in particular for DDoS attacks and worms. In order to accomplish the above task, the architecture of the system is based on two main ideas. First of all, to speed up the correlation of the raw alarms, exploiting the knowledge on the different types of attacks, the search is limited to specific “patterns”, where pattern means a subset of features. For instance, the features of the network flow, “source address”, “source port” and “destination port” should be sufficient to correlate raw alarms for a broad class of DDoS attacks. Moreover, in order to deal with the large number of raw alerts coming from the various IDSs of the network, the alert correlation system is implemented in a fully distributed form, based on a P2P protocol. Each member of the alert network is made up of a detection unit and a correlation unit. The detection unit is responsible for collecting raw alarms relative to the local traffic. These alerts are analyzed by the correlation unit that implements a multi-dimensional clustering algorithm. In particular, the traffic is preliminarily filtered according to a combinations of key features or patterns of different attacks previously described, the filtered

traffic is clustered and the traffic patterns are identified as suspicious by building a pattern lattice. Afterwards, the local results of this analysis are exchanged among the peers of the network for further analysis and correlation. The proposed system is tested on two data sets obtained from the Dshield.org website, which comprises a large number of firewall and NDIS logs collected from various platforms all over the world. In particular, one of the datasets employed in the testing contains data collected during the outbreak of the SQL-Slammer worm. The system is evaluated by conducting a large scale experiment on the PlanetLab network of the Princeton University in comparison to a fully centralized approach using metrics as detection accuracy and message exchange rate. The experiments demonstrates that the fully distributed approach is more efficient than the centralized approach in terms of the time required to correlate the alerts. Moreover, the above-mentioned probabilistic approach is really efficient in the case of stealthy attack scenarios.

In [14], a method to correlate alerts collected by different IDSs is presented. The IDSs inspect the local traffic by analyzing the content of the payload of HTTP requests. The analysis is based on the technique of  $n$ -grams analogously to the ANAGRAM algorithm[96]. The detection stage in the ANAGRAM system is improved following the technique of STAND (Sanitization Tool for ANomaly Detection) [26, 27] by a sanitization phase of the training set; then, the technique introduced in [28] is employed to tune the sensor parameters automatically. Once the payload is processed, the models associated with the normal traffic and with the suspicious packets are stored by using Bloom filters [13], which constitute their signatures. It is worth noting that storing the models of traffic by using Bloom filter has also the positive effect of preserving the privacy of the users during the correlation phase in which signatures are shared across different sites. Cross-site correlation of the alerts is obtained by using the method described in [68], i.e. the Worminator system. After that signatures are shared across sites, correlation is performed locally, by comparing the unencoded local alerts with the Bloom filter representing the alerts coming from remote systems. The system is evaluated on the datasets collected from the web servers of three different locations: `www.cs.columbia.edu`, `www.gmu.edu` and `www.cs.gmu.edu`. During the testing period the prototype was able to detect a considerable number of application-specific attacks previously unknown as well as a wide range of well-known attacks without human intervention. Finally, the system shows a very low rate of false positives.

In [51], another CIDS is presented. Its architecture is based on different local nodes, in which, during the training stage, the local network traffic is modeled by using an ensemble of gaussian mixture models. In practice, the ensemble is built by a variation of the AdaBoost algorithm, adapted to a streaming environment. Local parametric detection models are exchanged among the nodes and then they are combined on the local nodes in order to constitute a global model, by using a process based on particle swarm optimization and SVM to select the fusion function. Experimental tests, performed on the KDD'99 dataset, shows the effectiveness of the system in improving the detection accuracy, in spite of exchanging a relative small quantity of data among the nodes, i.e., the parameters that determine the gaussian mixture models.

It should be remarked that since only models of the traffic are shared and not network traffic itself, this guarantees the privacy of the various nodes involved.

An interesting point of view on the topic of correlating intrusion data coming from different nodes of a given subnetwork is proposed in [70]. The solution proposed relies on the concept of *conversation exchange model* introduced in [41]. This scheme has the scope of modeling the network traffic dynamic using aggregate traffic features and decision trees that encode, for each protocol, the way in which the packets are exchanged in the subnetwork considered. This process permits estimation of the distribution of the number of packets exchanged across the nodes of the subnetwork during nominal operations. At detection time, this distribution is monitored and techniques inspired by statistical mechanic and thermodynamics are used to detect anomalies in the distribution of the packets due to malicious activities. This approach is tested using different scenarios, i.e., different attacks on different segments of the network either occurring in different time slots or simultaneously. For each scenario, the system was able to detect the presence of attacks; however, it is worth remarking that the system detects deviations in the normal network dynamic. In order to detect the type of the attack or the nodes in which it has occurred, a deeper inspection at the level of packet content or of the network flow should be used.

In [78], the authors present a system to cluster malwares based on their behavior. Simple statistical features are extracted from the HTTP traffic generated by the malware, i.e., the length of the GET requests, the number of requests, etc. Then, the models of the malware behavior are clustered in a two stage procedure. In the first stage, the BIRCH ([100]) clustering algorithm is used, and its results are further refined by a hierarchical clustering. During the clustering process, cluster stability is performed by using the Davies-Boulding index in order to assess whether the clusters obtained are compact and well separated. Lastly, the signatures are generated from the clustering stages by using the Token-Subsequences algorithm [72]. Basically, for each cluster, signatures are generated considering substrings that are common to all HTTP payloads in that cluster. In [77], the same authors introduce some tweaks in the architecture, which presents a more balanced computational load and a better scalability in comparison with the original method. Experiments are conducted on many datasets of malware activities collected from various sources and the system proves to be able to generate signatures for different types of HTTP malwares.

#### 4.4.3 Data mining-based approaches

As there are not many data mining distributed implementation of intrusion detection systems, this section also includes some works based on data mining that are particularly suitable to be implemented in a parallel/distributed environment. A summary of the main works is reported in Table 4.2.

The work in [97] describes a framework for the problem of adaptive intrusion detection over unlabeled HTTP traffic streams, which follows the autonomic paradigm proposed by IBM [59]. Indeed, it is able to self-label the incoming data stream, to update continuously the detection model and moreover, it is capable of adapting the detection model after a change happens in the data stream. The IDS works by

Author(s)	Year	Technique	Correlation	Architecture	Dataset(s)	Evaluation
Zhou et al. [102]	2009	Raw alarms aggregation	Clustering	Fully distributed	Internet Storm Center Dshield.org	PlanetLab network of the Princeton University
Boggs et al. [14]	2011	$n$ -grams analogously to the ANAGRAM algorithm [96], models are encoded by Bloom filters	Exchanging models of traffic	Fully distributed	Real traffic from different sources	Real traffic from different sources
Hornig et al. [51]	2014	Ensemble of Gaussian mixture models	Particle swarm optimization, SVM	Fully distributed	KDD'99	None
McEachen et al. [70]	2007	Aggregate features	Conversation Exchange Model	Centralized	M.I.T. Lincoln Lab 1999	Appositely set up subnetwork
Perdisci et al. [78, 77]	2010, 2013	Statistical techniques	Hierarchical Clustering: BIRCH ([100])	Centralized	Malware samples	None

**Table 4.1.** Summary table of papers reviewed, CIDS. The column Evaluation indicates whether the framework is also evaluated on a real network.

inspecting HTTP packets at the payload level by essentially employing a 1-grams technique. After the preprocessing phase, in an initial stage, the data stream is clustered by an affinity propagation algorithm that employs a message passing technique named WAP (weighted affinity propagation). It is worth noting that clustering algorithms based on affinity propagation are graph-based and require a high computational load; therefore, they are generally more adapted to off-line analysis. However, the system adopts a message updating mechanism, which is suitable to be implemented in a distributed approach and that would extend its use to an online system. After the clustering phase is completed, the system analyzes the size and the sparseness of each cluster, in order to identify anomalous items. When a change is detected in the traffic, the models are rebuilt by restarting the clustering process. Three parameters control the rebuilding phase, i.e., the number of suspicious items, the length of the time window and the number of suspicious items since the last clustering stage. If one of these three parameters exceeds a certain threshold, the clustering phase is restarted. Finally, the system is compared with other existing clustering solutions: the  $k$ -Nearest Neighbor algorithm, the Principal Component Analysis ([56]), a one class SVM method and a version of the system based on the Sequential Karhunen-Loeve ([65]) transform. Testing is conducted on the KDD'99 dataset and on two large real HTTP traffic streams and the proposed system outperforms the other methods.

The approach in [21] presents an IDS (named UNIDS, *Network unsupervised intrusion detection system*) based on the misuse detection paradigm. The solution proposed, based on an ensemble clustering method, is suitable to be easily implemented on parallel/distributed architectures. In addition, although the authors test it on data stream coming from a single source, they point out it is well capable of dealing with data collected from different sources. The algorithm works in three phases: preprocessing, clustering and outlier identification. In the pre-processing phase, network traffic is captured from a single source using consecutive time windows of prefixed length. Afterwards, the captured traffic is aggregated into flows, and from each flow, some numerical features, i.e., source and destination network prefixes, traffic per time slot, etc. are extracted and a multi-dimensional vector representing the flow is obtained. A change detection algorithm is applied to the resulting vector and per-

mits to flag some flows in the time slot as anomalous, which are passed to clustering phase. Then, in the clustering phase, a sub-space clustering algorithm is applied to the data, in the following way. First, the flows are aggregated using either IPsrc or IPdst aggregation keys and each aggregated flow is described by a number of traffic attributes. Multidimensional data, obtained as described before, are projected on multiple 2-dimensional subspaces of the feature space and a density-based DBSCAN algorithm [34] is performed on each subspace by using different notions of similarity or distance (i.e., a simple Euclidean distance, but also more complex statistical-based similarities). Finally, the results of the different clustering applied in each subspace are combined by an *evidence accumulation clustering* method [43]; in particular, the algorithm of *Evidence Accumulation for Ranking Outliers (EA4RO)* is employed, better described in [21]. In the last phase, the top-ranked outlying flows are flagged as anomalies, using a simple thresholding detection approach.

Experimental results are conducted both on the well-known KDD'99 dataset and on traffic captured from two real networks. UNIDS is compared to traditional misuse based NIDs and it is comparable to the traditional approaches in its ability to detect unknown attacks; however, it is suitable to parallel computation, which permits a drastic reduction in the overall analysis time of the system.

An unsupervised approach based on clustering is introduced in [64]. The system adopts an adaptive grid algorithm, named fpMAFIA, based on the CLIQUE clustering technique [2], and its improvement pMAFIA described in [71]. The fpMAFIA algorithm basically employs *frequent-pattern tree* techniques in order to mine frequent item sets. This algorithm, based on the grid density clustering paradigm, starts by partitioning the features space in a grid, and part of the elaboration is basically performed by counting instances of the dataset in each element of the grid and establishing their relative density there, this involves the computation of the mean distance between two instances. Based on this elaboration grid elements are then merged to form clusters. The above analysis can then be implemented in parallel form by assigning, for instance, a certain number of grid elements to a single elaboration unit, e.g. a CPU core. The methodology proposed is tested on the KDD'99 dataset and it shows good performance in terms of elaboration speed and detection rate, but false positive rate is relatively high with respect to other methods proposed in the literature.

Huang et al. [52] propose an architecture based on an ensemble of on line sequential extreme learning machines, which is a type of neural network that supports on line sequential learning, i.e., the predictor is updated as soon as new instances become available, see [66] for more details. The approach combines different ensemble techniques such as Bagging, Subspace Partitioning and Cross Validation; in addition, in order to speed up the overall training phase, the data are distributed across the nodes of a cluster by means of the Hadoop framework. Experimental testing is conducted on the well-known KDD'99 dataset and the parallel implementation obtain a good scalability and a considerable reduction of the execution time and maintains the same accuracy as the sequential version of the approach.

Another distributed architecture based on neural networks is presented in Bukhtoyarov et al. [18]. In this system, a probabilistic approach (more detail on this tech-

nique are given in [19]) is used for the generation of a neural network on each node of a network. Then, the neural networks are combined in an ensemble, in which the combination function is evolved by employing genetic programming using the same technique presented in [19]. Afterwards, each node classifies the traffic independently from the other nodes and it inquires the ensemble only in the case in which it is not “confident” in its prediction. “Confidence” is implemented by a function based on the signal level at the output of a neuron corresponding to the class determined by the individual neural network classifier; if the value of this function is below a prefixed threshold, then the individual classifier needs to require the help of the ensemble. The system is evaluated on the KDD’99 dataset in term of Detection Accuracy and False Positive Rate against standard ensemble algorithms and in general it is capable of equal or better score than the competition with optimal choice of the above mentioned threshold for the confidence function.

The architecture in Brahmi et al. [15] combines misuse based and anomaly detection techniques with the multi agent paradigm. Indeed, the analysis of the traffic is subdivided among many agents and some of them are devoted to collecting and pre-processing the traffic; then, the traffic is filtered by the so-named misuse detection agent, which checks it by using some rules related to known attacks. Then, the filtered traffic is passed to the anomaly detection agent that analyze it by using a clustering algorithm that combines  $K$ -means and DBSCAN and identify normal and abnormal traffic. Finally, the result of this analysis is passed to the rule mining agent that, by using an association rule-based technique, builds new rules that will be used by the misuse detection agent. The system is implemented using the JADE (Java Agent DEvelopment Framework), and the agent platform and is suitable to be implemented in a distributed environment, however the experiments are conducted on a sequential version of the framework. Experimental testing is conducted on a real traffic dataset, in which the attacks are injected by the *Metasploit*<sup>1</sup> tool. The low bandwidth consumption and the network latency prove that the system is capable to cope with an increasing number of attacks and volume of network traffic. Moreover, the system is able to outperform largely used IDS systems, as SNORT, in term of detection rate and false positive rate.

#### 4.4.4 High-performance implementations

Recently, efforts have been made in order to take advantage of modern parallel hardware in the analysis of network traffic for the purpose of intrusion detection [95]. In fact, in the last few years, massively parallel hardware is readily available in the form of off the shelf components, namely multi-core CPUs and graphic hardware capable of general purpose computations (GPGPUs).

Most of the works present in the literature try to boost the signature verification stage by using a parallel implementation. For instance, there are projects that expand existing solutions in order to take advantage of GPGPU hardware, i.e., GNORT that is based on the well-known open source solution SNORT. Others works adopt a

---

<sup>1</sup> <http://www.metasploit.com/>

Author(s)	Year	Technique(s)	Dataset(s)	Preprocessing	Metric(s)	Implementation
Wang et al. [97]	2014	HTTP packet payload analysis based on 1-grams, WAP (Weighted Affinity Propagation) clustering	KDD'99, real traffic	None	ROC Curve	S
Casas et al. [21]	2012	DBSCAN algorithm [34], clustering ensemble: Evidence Accumulation for Ranking Outliers (EA4RO) ([21])	KDD'99, real traffic	Random Projections	ROC Curve, Detection Accuracy	S
Leung et al. [64]	2005	Grid density clustering [71]	KDD'99	None	ROC Curve	S
Folino et al. [38]	2010	Genetic Programming, Ensemble of Decision Trees	KDD'99	None	Detection Rate, False Positive Rate, Detection Accuracy	D
Folino et al. [40]	2016	Genetic Programming, Meta-Ensemble of Decision Trees	KDD'99, real traffic	None	Detection Rate, False Positive Rate, Detection Accuracy	D
Huang et al. [52]	2016	Ensemble of on line sequential extreme learning machines	KDD'99	None	Detection Accuracy	D
Bukhtoyarov et al. [18]	2014	Genetic Programming, Neural Networks	KDD'99	None	Detection Accuracy, False Positive Rate	D
Brahmi et al. [15]	2011	Multi Agent System, DBSCAN clustering, association rule mining	Real traffic	None	Detection Rate and False Positive Rate	S

**Table 4.2.** Summary table of papers reviewed, data mining distributed (In the last column (Implementation): D indicates Distributed implementation, S indicates Suitable to distributed implementation)

signature-based solution from scratch, and concentrate their efforts on implementing a very efficient signature verification based on parallel pattern matching algorithms [55, 7]. This approach can be extended by distributing the signature verification among different nodes. In this case, a particular attention is devoted to the problem of synchronizing the elaboration between nodes, see [92, 55, 94, 82].

An interesting work is presented in [6], in which GPGPU hardware is paired to a cluster. In particular, computational tasks relative to signature matching are performed by the GPGPUs. The cluster is employed to process and mine the data streams of the alarms, and to assess the most common vulnerabilities and threats, by exploiting the advantages of the Hadoop framework.

In order to take full advantage of a parallel implementation, a key point is to guarantee the scalability and a good load balancing. To this aim, the system proposed in [101] adopts a solution called negative pattern matching, which splices the preprocessed network traffic into small segments suitable to be analyzed in parallel and to perform a fine grained load balancing of the computational load. This technique ensures that no key information is lost in the process. Moreover, the rule set is subdivided into independent subsets, which can be applied independently, providing another means to parallelize the pattern matching process. Finally, the pattern matching is performed by a technique based on the ternary content addressable memory method. Experimental results show that the above-mentioned techniques are able to improve remarkably pattern matching performance and scalability in many real-world scenarios.

The approach in [3] uses an anomaly-based technique, which adopts a parallel local outlier algorithm. The technique used comprises a heavy phase of computation of the mutual distances of the points in some regions of the dataset. Therefore, a parallel implementation that increases the speed of the algorithm can be set up, as the computation of the distances can be performed independently. It would be in-

teresting to implement more sophisticated solutions on parallel hardware and to test the systems on traffic data captured and processed in real time. For instance, modern database systems such as Oracle Database 10g (see [20]) provide all the necessary instruments to easily implement an intrusion detection architecture, from the data preprocessing to the phase of clustering and analysis. The possibility of exploiting the potentialities of the Oracle grid computing infrastructure is really interesting. In fact, this infrastructure enables sharing the content of the databases, or in other words the network traffic to be analyzed using the power capacity of many computing nodes. Therefore, it is possible to distribute the computational load in a transparent way and to ensure the availability of flexible on-demand computing resources capable of guaranteeing scalability and reliability. This infrastructure could provide a good testbed to compare different solutions and techniques.



## **An architecture for combining ensemble of classifiers**

In this Chapter the proposed architecture for combining an ensemble of classifiers is presented. First of all, a definition of the problem and the description of the ensemble approach is presented along with the definition of the components of the system: the base classifiers, the combination function, the tool to build the combination function and the working aspects of the approach.

Then, the details of some possible scenarios apt to be exploited with this approach are described.

The first scenario concerns the problem of the user profiling, i.e. to identify group of users that exhibit the same behaviors. The problem is presented in Chapter 2.1. Afterwards, a version of the algorithm optimized to work with missing data is presented.

The user profiling is a common scenario in the cyber security field that deals with missing data problem, but others cases could be addressed by the framework, for instance the problem of unbalanced classes. This topic is very related with intrusion detection task. Considering that the classes relative to attacks are very "uncommon" compared with normal instances, it is easy to model the problem of detecting attacks as a problem of unbalanced data.

Thus, a solution to handle unbalanced classes is proposed and the architecture of an intrusion detection system based on an ensemble of classifiers is presented. The main goal of the proposed architecture is the detection of different type of attacks in a distributed environment. Moreover, a desired behavior is that the model should be updated during execution and it should work with streaming data. The last part of this Chapter presents solutions to achieve these goals. First, a general architecture for an IDS is defined along with some preprocessing procedure to explain how the IDS works. Then the description of the algorithm in the case of static data is presented. In this context, the term 'static data' is referred to the traditional approach of the classification where the model is builded on a training dataset then it is used to classify new instances. The other case addressed is the streaming data problem where the model must be updated to handle new concepts observed in the stream.

In the latter case, the algorithm is extended with a module to handle different types of drift. The combination of ensembles and drift detection algorithms produces

a system able to update its models with the advantages of a major flexibility in detecting new attacks. To improve such type of applications, the Section 5.4 presents a variant using specialized classifiers, each one devoted to a specific type of attack. Finally, the general architecture of an intrusion detection system based on the ensemble paradigm is provided.

## 5.1 The technique for combining the ensemble of classifiers

Although, the Section 3 provides the background of the techniques used in the framework, the main concepts are summarized also in this Section and the implementation details of building an ensemble are provided. The general schema for combining an ensemble of classifiers is shown and can be specialized for a particular application. It can be used in order to combine an ensemble of classifiers without the need of a further phase of training. Then, the distributed GP framework used to evolve the combining function of the ensemble is illustrated.

### 5.1.1 The combining function

The framework is based on schema shown in Figure 3.1, in which different algorithms are used on the same dataset in order to build the different classifiers/models and the formalization of this approach is described in the following.

The result of the process is to obtain the models of the ensemble's classifiers and the combination function used to combine their prediction.

Let  $S = \{(x_i, y_i) | i = 1, \dots, N\}$  be a training set where  $x_i$ , called example or tuple or instance, is an attribute vector with  $m$  attributes and  $y_i$  is the class label associated with  $x_i$ . A predictor (classifier), given a new example, has the task to predict the class label for it.

The work of the algorithm is to build  $g$  predictors, each on a different training set, then combine them together to classify the test set. As an alternative, the  $g$  predictors could be built using different algorithms on the same/different training set.

The combination function is based on the schema shown in Figure 3.1: it does not need any further phase of training, whether the functions used can be combined without using the original training set.

The combination function combines the support computed by each base classifiers. The support is the probability that a tuple belong to a certain class according to the prediction ability of the classifier. The formal definition is described in the following.

Let  $x \in R^N$  be a feature vector and  $\Omega = \{\omega_1, \omega_2, \dots, \omega_c\}$  be the set of the possible class labels. Each classifier  $h_i$  in the ensemble outputs  $c$  degrees of support, i.e., for each class, it will give the probability that the tuple belongs to that class. Without loss of generality, all the  $c$  degrees are in the interval  $[0, 1]$ , that is,  $h_i : R^N \rightarrow [0, 1]^c$ . Denote by  $H_{i,j}(x)$  the support that classifier  $h_i$  gives to the hypothesis that  $x$  comes from class  $\omega_j$ . The larger the support, the more likely the class label  $\omega_j$ . A non-trainable combiner calculates the support for a class combining the support values

of all the classifiers. For each tuple  $x$  of the training set, and considering  $g$  classifiers and  $c$  classes, a Decision Profile matrix DP can be build as follow:

$$DP(x) = \begin{bmatrix} H_{1,1}(x) & \dots & H_{1,j}(x) & \dots & H_{1,c}(x) \\ H_{i,1}(x) & \dots & H_{i,j}(x) & \dots & H_{i,c}(x) \\ H_{g,1}(x) & \dots & H_{g,j}(x) & \dots & H_{g,c}(x) \end{bmatrix}$$

where the element  $H_{i,j}(x)$  is the support for  $j$ -th class of  $i$ -th classifier.

The functions used in the approach simply combine the values of a single column to compute the support for  $j$ -th class and can be defined as follow:

$$\mu_j(x) = F[H_{1,j}(x), H_{2,j}(x), \dots, H_{g,j}(x)]$$

The class label of  $x$  is the class with maximum support  $\mu$ .

### 5.1.2 A distributed tool to evolve the combiner functions

The combining function presented above needs a tool able to compute it. The tool used in this work is a distributed/parallel GP implementation, named CelluAr Genetic programming (CAGE) [36], running both on distributed-memory parallel computers and on distributed environments.

#### Functions, terminals and fitness evaluation

The model that the GP system uses in order to combine the predictions of multiple base classifiers is presented in background section. This Section describes the implementations of nodes and terminals of the combining function.

Differently from classical models in which the GP tool is used to evolve the models, in this approach, the classifiers (with an associated weight previously computed on the training set) are the leaves of the tree, while the combiner functions are placed on the nodes. A combiner function is an algebraic function to combine probabilities. In particular, the functions chosen to better combine the classifiers composing the ensemble are non-trainable functions and are listed in the following: average, weighted average, multiplication, maximum and median. They can be applied to a different number of classifiers, i.e. each function is replicated with a different arity, typically from 2 to 5.

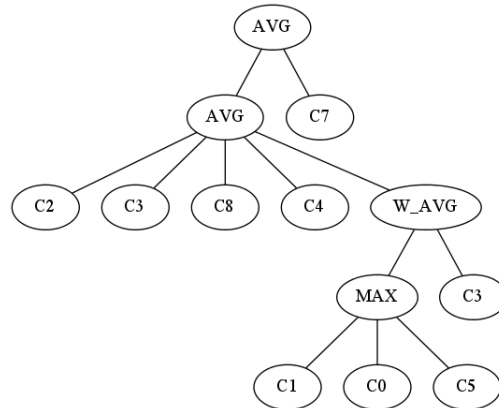
The **average** function, used with an arity of 2, 3 and 5, is defined as:  $\mu_j(x) = \frac{1}{g} \sum_{i=1}^g H_{i,j}(x)$ .

The **multiplication** function (arity 2, 3 and 5) is defined as:  $\mu_j(x) = \prod_{i=1}^g H_{i,j}(x)$ .

The **maximum** function returns the maximum support for 2, 3 and 5 classifiers and can be computed as:  $\mu_j(x) = \max_i \{H_{i,j}(x)\}$ .

The **median** function (arity 3 and 5) can be computed as:  $\mu_j(x) = \text{median}_i \{H_{i,j}(x)\}$ .

Finally, the **weighted** version of the **average** function uses the weights computed during the training phase to give a different importance to the models on the basis of the performance on the training set, and can be computed as:  $\mu_j(x) = \frac{1}{\sum_{i=1}^g w_{i,j}} \sum_{i=1}^g w_{i,j} * H_{i,j}(x)$ . For this function the values of 2, 3 and 5 are chosen for the arity.



**Fig. 5.1.** An example of GP tree generated from the tool.

In order to better clarify, how the tree is built, in Figure 5.1, an example of tree generated from the tool is illustrated. The GP model requires the computation of a fitness function, i.e. an objective function that give informations about the distance of the current solution from the goal to achieve. In this formulation the fitness function is simply computed as the error of the ensemble on the validation set, i.e. the ratio between the tuples not correctly classified and the total number of tuples. A solution with an error close to 0 is the ideal result.

### Unbalanced data

However, in the particular case of unbalanced datasets, a weighted fitness is adopted. When the data have unbalanced classes the difference between instances of minority classes and other classes is measured in orders of magnitude. Then, an error on a minority class has a very little weight compared to the overall performance. In this case the standard fitness function used to compute the model produces a classifier that penalize the minority class. To overcome this issue a modified fitness function is used. In practice, if a tuple belonging to a minority class is misclassified, the fitness function is penalized by a weight equal to the ratio between the total number of tuples and the total number of tuples belonging to that class (to avoid really high weights, if the weight exceeds the threshold value of 10, it is fixed to this threshold). For the tuple belonging to the majority class, the penalty weight is fixed to 1, as in the case of balanced datasets. In this way, like a cost-based approach, errors on minority class have a greater importance than errors on other classes and the final classifier is able to address successfully the unbalanced data problem.

### The main algorithm used for computing the combining functions

The function defined above is computed by a tool based on a fine-grained cellular model. The overall population of the GP algorithm is partitioned into subpopulations

of the same size. Each subpopulation can be assigned to one processor and a standard (panmictic) GP algorithm is executed on it. Occasionally, migration process between subpopulations is carried out after a fixed number of generations. For example, the  $n$  best individuals from one subpopulation are copied into the other subpopulations, thus allowing the exchange of genetic information between populations. The model is hybrid and modifies the island model by substituting the standard GP algorithm with a cellular GP (cGP) algorithm. In the cellular model each individual has a spatial location, a small neighborhood and interacts only within its neighborhood. The main difference in a cellular GP, with respect to a panmictic algorithm, is its decentralized selection mechanism and the genetic operators (crossover, mutation) adopted.

This tool is used to evolve the combiner functions and obtain an overall combiner function, which the ensemble will adopt to classify new tuples. Implicitly, the function selects the classifiers/models more apt to the particular datasets considered.

To summarize, considering a dataset partitioned in training, validation and test set, the approach works using the following steps.

1. The base classifiers are trained on the training set; then, a weight, proportional to the error on the training set, is associated to each classifier together with the support for each class, i.e. the decision support matrix is built. This phase could be computationally expensive, but it is performed in parallel, as the different algorithms are independent from each other.
2. The combiner function is evolved by using the distributed GP tool, CAGE, on the validation set. No extra computation on the data is necessary, as validation is only used to verify the correct class is assigned and consequently to compute the fitness function.
3. The final function is used to combine the base classifiers and classify new data (test set). This phase can be performed in parallel, by partitioning the test set among different nodes and applying the function to each partition.

## 5.2 The missing data problem and the framework solution

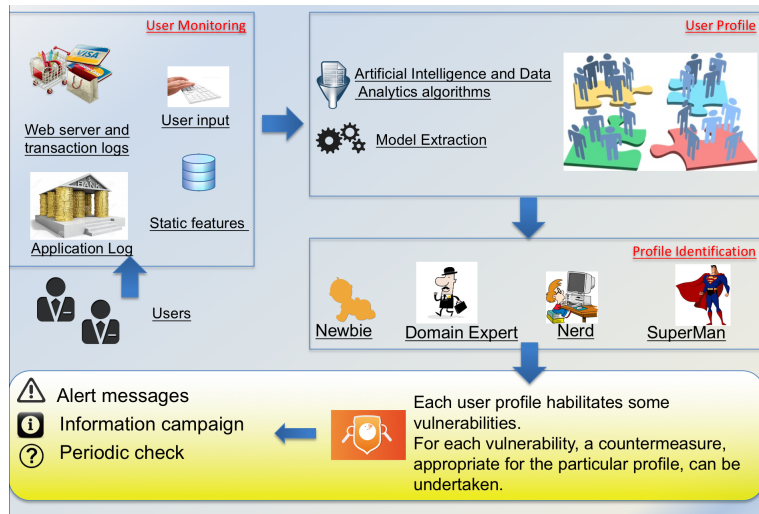
The first issue addressed designing the framework is the missing data problem. This Section illustrates the software architecture designed to deal with this issue and the cyber security scenario presented in Section 2.1 is detailed; then, it is presented the description of mechanism used by the distributed GP framework to evolve the combining function of the ensemble in the case of missing data.

### 5.2.1 An architecture for classification of user profiles in e-payment systems

A special case of missing data problem is the scenario presented in Chapter 2.1. In the following an architecture specialized for classification of user profiles in e-payment systems is presented.

In the scenario of classification of user profiles, the classes, in which the users will be divided, are individuated on the basis of their expertise in computer science

and in the domain of the e-payments systems. Indeed, most of the vulnerabilities are associated with the behavior and the practices correlated with the knowledge of the computer and/or of the e-payment system. As example, contrarily to the normal belief, a vulnerability study confirmed that software developers are the most vulnerable to attacks [88]. Furthermore, an excess of confidence and the consequent download and installation of a number of applications can cause vulnerabilities; in the same way, misconfigurations of the system due to inconsistent application of security associated with a lack of competency could abilitate other kinds of vulnerabilities. Similar behavior could be seen in the activities of users of the e-payments system.



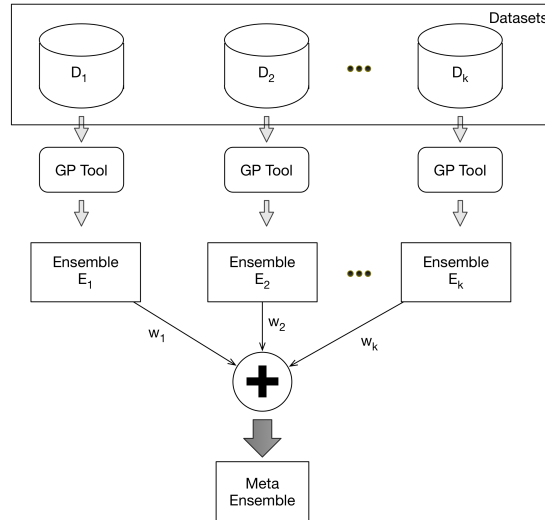
**Fig. 5.2.** A general architecture to monitor and to classify the users of an e-payment system and to undertake the needed countermeasures.

Given these considerations, Figure 5.2 shows the general architecture designed to handle the problem of mitigating the consequences of the user behavior. The information concerning the user is supplied by using different sources of information or monitoring tools (i.e. generally automatic software analyzing the action and the behavior of the users). Going more into detail, user datasets can include demographic and education information, e.g., name, age, country, education level, computer knowledge, task knowledge, etc. and may also includes information concerning the contest in which the users operate and the roles they have in the systems. In addition to these data, which usually do not change if we consider a reasonable amount of time, the monitoring tool collects operational and behavioral data (e.g. IP addresses from which users connect to the system, operating system and browser used, the duration of the session, etc.), for which changes over time should be also considered. Finally, we also collect user input (i.e., commands entered using the keyboard or via GUI, using the mouse, etc.) This information should be captured in a dynamic way,

by logging user actions. Unfortunately, all these kinds of data are not present for each user for clear reasons of privacy and for a number of different motivations (i.e., we have users with different roles and therefore, it is possible to monitor only some types of user, some users do not want to give authorization to disclose some data, etc.). Therefore, for different users, some sources are missing and this problem must be faced efficiently in order to obtain an accurate classification.

**5.2.2 The software architecture of the meta-ensemble approach.**

The architecture presented in Section 5.2.1 is exploited with the details about the scheme of the combination of the base classifiers. The Figure 5.3 shows the software architecture of the meta-ensemble approach adopted to analyze the first scenario introduced in Section 2.1. Note that CAGE-MetaCombiner is able to work on incomplete datasets (named  $D_1, D_2, \dots, D_k$  in the figure). It is worth noticing that, as described in the background section, it is equivalent whether each dataset comes from a different source of data, or they are obtained from a partition of an incomplete dataset by removing groups of missing features. The only strong assumption is that each corresponding tuple of the different datasets is used to predict the same class. The corresponding tuple can be missing in one or more datasets, but if it is missing in all the datasets, it will be discarded and counted as a wrong prediction in the evaluation phase.



**Fig. 5.3.** The software architecture of the meta-ensemble architecture.

In practice, an ensemble is built for each dataset by using a distributed GP tool, CAGE (better described in the next subsection), to generate the combiner function

(see Figure 5.1 for an example). The learning models (classifiers) composing the ensemble are taken from the well-known WEKA tool (see subsection 6.3.1 for more details on the algorithms used). The different ensembles perform a weighted vote in order to decide the correct class. It is worth remembering that each ensemble evolves a function for combining the classifiers, which does not need any extra phase of training on the original data. The final classification is obtained computing the error using the same formulae as the Adaboost.M2 algorithm used by the boosting algorithm, by computing the error of the entire ensemble instead of a single classifier as in the original boosting algorithm.

The entire process is better detailed in the pseudocode in Figure 5.8. We consider  $l$  base classifiers and  $k$  incomplete datasets. Each incomplete dataset is partitioned into train, validation and test set. A number of classification algorithms were trained on the training sets and only the best  $l$  (a predefined threshold) are maintained and take part in the ensemble. Then, a decision profile matrix is built for each classifier in order to optimize the subsequent phase, in which the GP tool evolves the combiner function of the ensemble, by using the validation set. A weight is associated with each ensemble on the basis of the error of the ensemble on the validation set.

Afterwards, for each tuple  $x$ , for each possible class  $j$  and for each ensemble  $i$ , the errors are computed using a weighted mean:  $\mu_j(x) = \frac{\sum w_i * E_{ij}(x)}{\sum w_i}$  where  $w_i$  is the weight of ensemble  $E_i$  computed in the validation phase and the term  $E_{ij}(x)$  is the probability that the tuple  $x$  belongs to class  $j$  returned by ensemble  $i$ . The weight is equal to the error rate of the ensemble computed on the classification of the tuples of the validation dataset. The purpose of associating a weight to an ensemble is to penalize/reward the ensembles having best/worse performances and also reducing the impact of classification errors.

The final classification is obtained by using the formula  $class(x) = argmax_j(\mu_j(x))$ ; if a tuple  $x_i$  is missing, the corresponding ensemble  $E_i$  is discarded.

```

Let  $\alpha$  be the total number of base classification algorithms.
Let  $l$  be the number of base classification algorithms effectively used.
Given a set of  $k$  incomplete datasets  $D_1, D_2, \dots, D_k$ .
where a dataset  $D_i = \{X_{i1}, X_{i2}, \dots, X_{im_i}\}$  and  $X_{ik}$  is a tuple  $\{A_1, A_2, \dots, A_d, C\}$ 
where  $A_j$  is an attribute and the class  $C$  can have  $c$  possible values.
Note that each tuple  $X_{ik}$  can potentially be missing.

For each  $D_i$ 
  Consider the dataset  $D_i$  partitioned into train, validation and test set:  $D_{train_i}, D_{valid_i}$  and  $D_{test_i}$ 
  Train  $\alpha$  different classification algorithms on  $D_{train_i}$ 
  Maintain the  $l$  classifiers obtaining the highest accuracy on the training set.
  Build  $l$  decision profile matrices, one for each of the classifiers,  $DP_1, DP_2, \dots, DP_l$ 
  using the respective validation set, one for each classifier of dimension  $|D_{valid_i}| \times c$ 
  Run the distributed GP tool on the validation set  $D_{valid_i}$  in order to obtain
  the combiner function of the ensemble.
  Obtain an Ensemble  $E_i$ , a combiner function  $F_i$  and a weight  $W_i$ ,
  computed on the basis of the error of the given ensemble on the validation set.
end for each
Build the decision profile matrix  $DP$  of the entire ensemble  $E$ ,
where each element  $H_{i,j}(x)$  is the support that classifier  $h_i$  gives to the hypothesis
that the tuple  $x$  comes from class  $j$ .
Compute the weighted mean for each class  $j$ :  $\mu_j(x) = \frac{\sum w_i * H_{ij}(x)}{\sum w_i}$  on the test set.
Compute the class by using the formula  $class(x) = argmax_j(\mu_j(x))$ .
Note that if a tuple  $X_{ik}$  is missing, the corresponding ensemble  $E_i$  does not participate to the voting procedure.

```

Fig. 5.4. The pseudo-code of the algorithm.



### 5.3 A distributed framework for intrusion detection

In the following, it is outlined a specialization of the general architecture presented in the previous sections and adapted for the detection of network intrusion and it is illustrated the particular preprocessing phase adopted.

#### 5.3.1 Preprocessing methodology

Usually network traffic is acquired with the aid of software based on `libpcap`, such as `tcpdump`, and stored in the standard `pcap` (packet capture) format. Other software alternatives for this task include `Gulp` more apt to an high performance environment more apt to an high speed network environments.

A dataset of network connections consists of standards PCAP (packet capture) files, one for each day containing the relative network traffic. Network data streams acquired in form of PCAP files require a preprocessing phase in order to extract features needed in the subsequent analysis. With respect to this task, network traffic may be modeled at the level of flows, i.e., first the connections are aggregated into flows and then suitable features are extracted and aggregated. Beside the level of flows, data streams may be analyzed at the level of packet payloads, using the so-named *deep packet inspection* technique. Indeed, the payload of HTTP packets is made up of textual content and then it may be analyzed by using the technique of *n*-grams, usually employed for the analysis of textual content. It is worth to notice that *n*-grams based techniques have the drawback of presenting an exponential growth in *n* of the feature space and consequently they are computationally intensive. In any case, note that the choice of techniques used to analyze the data streams depends clearly on the type of analysis performed in the subsequent stages. In order to extract a relatively compact set of features, the choice is to model the traffic at the level of flows; in practice, the `pcap` files are aggregated in flows and preprocessed with the aid of the `flowcalc`<sup>1</sup> tool from the MuTriCs project, Fomerski et al. [42].

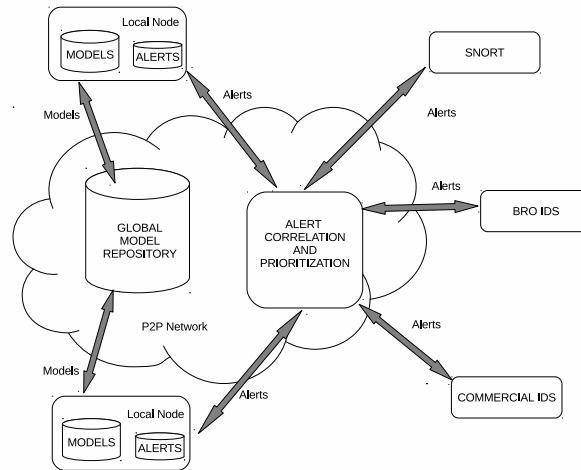
This tool is preferred over more common choices such as `tcptrace` since it is more apt to a streaming environment. Moreover, it can be extended with a number of plugins; for instance, the plugins which permit to process the payload and to extract basic statistics from it. In particular, in this work the following plugins are enabled: `basic`, `counters`, `pktsize`, `lpi`, `web`. These modules provide respectively basic statistics on packets payload such as size and inter arrival times, number of packets and bytes, sizes of the first packets in the flow, payload protocol and finally various statistics on the web traffic content.

#### 5.3.2 The architecture of the IDS

In this Section the specialization of the framework to deal with the detection of the intrusions is presented. The proposed architecture is build around the Collaborative Intrusion Detection paradigm, i.e. it implements a mechanism that correlates

<sup>1</sup> <http://mutrics.iitis.pl/flowcalc>

information coming from different nodes or sensors in which network data stream is collected. This mechanism permits the discovery of more complex attacks and may work either at the level of alarms or at the level of models generated for a particular type of attack. Sharing models of malicious activity has the side effect of spreading the knowledge acquired locally to other nodes; in practice, a node can receive models for attacks observed elsewhere, but not locally and use them in a proactive way to inspect the local traffic. Moreover, working at the level of alarms, has the advantage of enabling collaboration among nodes of different nature, i.e. running commercial IDSs as well as open source solutions such as Snort and BroIDS to share meaningful information, see Figure 5.5.

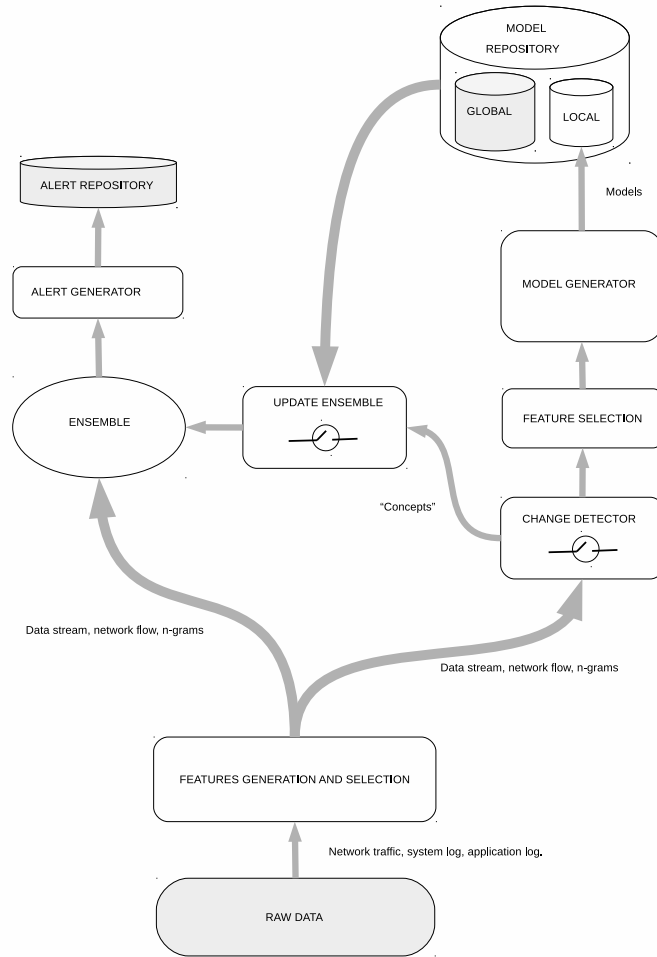


**Fig. 5.5.** Ensemble-Based NIDS General Architecture

For the sake of scalability, a distributed alert correlation mechanism that involves each single node as active part is illustrated in the architecture.

In order to accomplish the above tasks, and also to cope with the fast changing nature of network traffic and attacks, the local components of the architecture contains some specialized interacting modules. In fact, the module Change Detector is responsible for a large scale analysis of the data stream that flows through a local node in search of significant deviations from the normal behavior. In order to guarantee real time responses, the Change Detector analyzes the data stream in time windows and subsequently it computes suitable functions on values obtained from aggregated features extracted from each time window; we call the set of values of these functions, obtained from a window, a “concept” (relative to that time window).

Many algorithms, based for instance on time series or fractal dimension, can be employed to detect anomalies in the sequence of “concepts”, and, in the case anomalies are detected, the Model Generator is activated consequently. It builds new mod-



**Fig. 5.6.** A general architecture for classifying large streams of attacks and normal connections.

els by classification, clustering and by using other statistical methods and maintains the model together with the relative “concept”. The overall ensemble is employed to monitor the network data stream, see Figure 5.6. Since it is not plausible to add models indefinitely to the ensemble, a module called Update Ensemble is responsible for managing the active models of the ensemble; a strategy, currently in development, permits to compare “concepts” stored as metadata in the models with that coming from the current data stream, and consequently activate/deactivate the models.

As already said, the ensemble incorporates models of traffic and it is responsible for the analysis of the incoming data stream in search of malicious activities.

IDSs benefit from using algorithms based on the ensemble paradigm for a number of reasons. Indeed, given the speed of modern networks and the consequent increase in the volume of data that need to be processed, solutions based on algorithms that can be easily implemented in parallel/distributed environment (multi-core CPUs and GPGPU hardware) should be preferred, and clearly solutions based on the ensemble paradigm can be easily implemented on parallel hardware. Moreover, models may be trained on different type of attacks or on some parts or on some levels of the network and finally combined together, to assure a better prediction. The following section provides the implementation description of the model generator module based on a evolutionary approach and the meta-ensemble technique used in the framework.

#### 5.4 A meta-ensemble approach for combining specialized ensemble of evolved classifiers

In this Section, the approach is adapted to combine specialized classifier. Each model of the ensemble is created to handle a specific type of attacks or a particular class. In this way, the classifier is able to adapt its behavior to different distribution of connections, also the model can be shared between different nodes of a network without reducing the ability to detect attacks.

This approach can be seen as a preliminary work to extend the general framework to a new class of intrusion detector: the final goal is to design a system suitable to process data stream and it can adapt/update its models to changes of the data.

The model generator of the distributed architecture shown in section 5.5 is built on the well-known Massive Online Analysis (MOA) toolbox<sup>2</sup>, adopted to handle the data streams and also used for the implementation of the classifier algorithms and on the distributed GP tool, CelluAr GEnetic programming (CAGE) [36], which is used to evolve the combining function of the ensemble. The latter is based on the fine-grained cellular model and runs both on distributed-memory parallel computers and on distributed environments, by partitioning the overall population of the GP algorithm into subpopulations, one for computation node; then a standard (panmictic) GP algorithm is executed on each node.

As the streams flows, a common supposition is required to run the software: that part of input data is labelled. When a sufficient number of labelled tuples is collected, they are divided into training set and validation set and used to train the ensemble, as described in the following. The base classifiers, chosen among the best performing on a set of benchmarks of the MOA toolbox, are trained on the training set; then, a weight, proportional to the error on the training set, is associated to each classifier; at the same time a decision support matrix is built. This phase could be computationally expensive, but it could be performed in parallel, as the different algorithms are independent from each other.

---

<sup>2</sup> <http://moa.cms.waikato.ac.nz/overview/>.

After that, the combiner function of the ensemble is evolved by using the distributed GP tool, CAGE, on the validation set. As nodes of the GP tree, some non-trainable functions are chosen, while the leaves of the tree are the different classifiers selected in the previous phase (see Figure 5.1) and the fitness function is simply the accuracy of the ensemble computed on the validation set. It is worth to remember that no extra computation on the data is necessary, as the validation set is only used to verify the correct class is assigned and consequently to compute the fitness function.

Finally, the overall combiner function is used to classify the new coming tuples of the stream. Implicitly, the function selects the classifiers/models more apt to the particular datasets considered. The final function is used to combine the base classifiers and classify the incoming stream. Also this phase can be performed in parallel, by partitioning the stream among different nodes and applying the function to each partition.

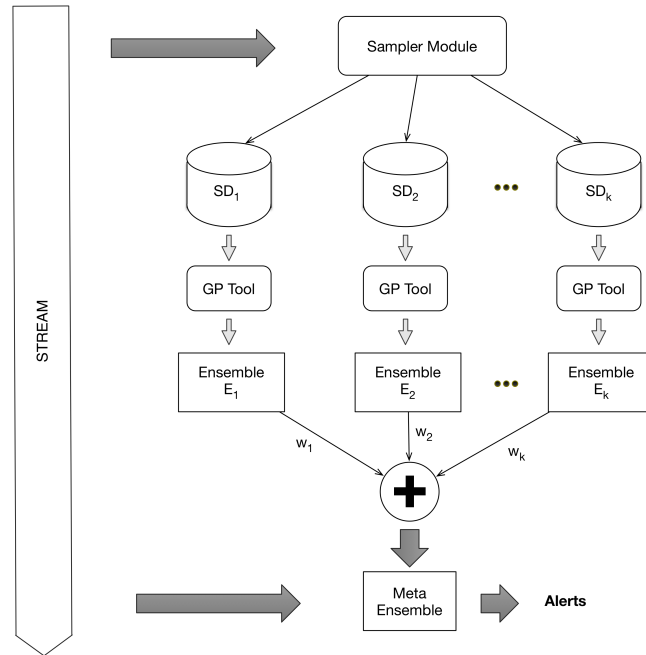


Fig. 5.7. The software architecture of the meta-ensemble architecture.

The entire process is better illustrated in Figure 5.7 and then specified in the pseudocode of Figure 5.8. An infinite stream of tuples is supposed to flow as input to the model generator and only part of them are labelled. In the startup phase, in which the ensemble is built, each newly arrived tuple, i.e.  $T_i$ , is passed to the classifier that assigns a class to it. The main idea of this approach is adopt a meta-ensemble paradigm and to specialize each ensemble on a particular type or set of attacks. To

this aim, a sampler module extracts the labelled tuples coming from the streams and assigns them to the appropriate datasets (together to a corresponding percentage of normal connections, 50% in our experiments), named  $SD_1, SD_2, \dots, SD_k$ . Then, an ensemble for each specialized dataset is built, by using the distributed GP tool.

Finally, the different ensembles perform a weighted vote in order to decide the correct class. It is worth to remember that each ensemble evolves a function for combining the classifiers, which do not need of any extra phase of training on the original data. The final classification is obtained computing the error using the same formulae of the Adaboost.M2 algorithm used by the GP tool, by computing the error of the entire ensemble instead of a single classifier as in the original boosting algorithm.

Afterwards, for each tuple  $x$ , for each possible class  $j$  and for each ensemble  $i$ , the errors are computed using a weighted mean:

$$\mu_j(x) = \frac{\sum w_i * E_{ij}(x)}{\sum w_i}$$

an the final classification is obtained by using the formula

$$class(x) = argmax_j(\mu_j(x))$$

<p>Let <math>\alpha</math> be the total number of base classifiers algorithms.  Let <math>l</math> be the number of base classifiers algorithms effectively used.  Given a set of <math>k</math> datasets <math>SD_1, SD_2, \dots, SD_k</math>.</p> <p><b>For each <math>D_i</math></b>      Consider the dataset <math>SD_i</math> partitioned in train, validation and test set: <math>SD_{train_i}, SD_{valid_i}</math> and <math>SD_{test_i}</math>      Train <math>\alpha</math> different classification algorithms on <math>SD_{train_i}</math>      Maintain the <math>l</math> classifiers obtaining the best accuracy on the training set.      Build <math>l</math> decision profile matrixes, one for each classifiers, <math>DP_1, DP_2, \dots, DP_l</math> using the respective validation set, one for each classifier of dimension <math>n \times c</math>, where <math>n</math> is the number of tuples and <math>c</math> the number of classes.      Run the distributed GP tool on the validation set <math>SD_{valid_i}</math> in order to obtain the combiner function of the ensemble.      Obtain an Ensemble <math>E_i</math>, a combiner function <math>FC_i</math> and a weight <math>W_i</math>, computed on the basis of the error of the given ensemble on the validation set.</p> <p><b>end for each</b>  Build the decision profile matrix <math>DP</math> of the entire ensemble <math>E</math>, where each element <math>H_{i,j}(x)</math> is the support that classifier <math>h_i</math> gives to the hypothesis that the tuple <math>x</math> comes from class <math>\omega_j</math>.  Compute the weighted mean for each class <math>j</math>: <math>\mu_j(x) = \frac{\sum w_i * H_{ij}(x)}{\sum w_i}</math> on the test set.  Compute the class by using the formula <math>class(x) = argmax_j(\mu_j(x))</math>.</p>
--

**Fig. 5.8.** The overall algorithm of the distributed IDS.

## 5.5 The framework and the drift algorithms

The general architecture of the IDS presented in the previous Section is extended to be applied in another context, i.e data streaming, and a new module is added to the

framework, a change detector based on well-known drift algorithms. In this case, the strategy adopted is to update/replace base models of the ensemble when a "change" is detected.

Referring to the Figure 5.5, in which the general architecture for an innovative intrusion detection system is described, in this Section the focus is on the Change Detector Module. Typically, the stream of data, that is the input of the framework, can originate from different sources: network traffic coming from a particular interface or coming from a router, system logs, application logs concerning the software installed on the system, etc. After the preprocessing phase, the features are extracted and the data stream is ready for further analysis. The stage of the elaboration of the stream is performed by the module called Change Detector. It performs an analysis of the data stream seeking qualitative deviations from the normal behavior. Indeed, as it must perform a real time analysis, it typically divides the data stream in time windows of prefixed duration and then some functions are computed on the values of aggregated features coming from the window considered. The set of the values of these functions captures qualitative characteristics of the data stream; if an anomaly is detected in these sequences, or in the initial training phase of the system, the module Model Generator is activated in order to generate new models for the analysis of the data stream, performed by the Distributed Ensemble module. These new models are used to update the ensemble by using some replacement strategies or simply by adding them to the current ensemble. Finally, the overall ensemble is used to classify the new incoming tuples and to generate some alerts to be processed in some way.

Following the schema illustrated in Subsection 5.5, the framework, which must operate in order to quickly detect and manage drift, must be composed by some main modules: a drift (change) detector, a generator of classifiers (models), a module to replace old/inefficient/overlapping classifiers. In addition, the framework also includes a module to generate the combiner function of the ensemble by using a genetic programming approach, in the same way described before.

The combining function is computed with the same approach described in 5.1.2. This tool is used to evolve the combiner functions and obtain an overall combiner function, which the ensemble will adopt to classify new tuples. Implicitly, the function selects the classifiers/models more apt to the particular datasets considered. Briefly, as nodes of the GP tree, some non-trainable functions are chosen, better specified in the experimental section, while the leafs of the tree are the different classifiers selected in the previous phase (see Figure 5.1) and the fitness function is simply the error of the ensemble computed on the validation set.

The model generator is built on the well-known Massive Online Analysis (MOA) toolbox, adopted to handle the data streams and also used for the implementation of the classifier algorithms and for the drift detection strategies described in the Subsection 5.5.1. Finally, the module to replace old/inefficient/overlapping classifiers adopts the strategies also described in the Subsection 5.5.1.

The way in which the overall algorithm works is better illustrated in the pseudocode of Figure 5.9. In current implementation an hypothesis is made: the detection method uses a window of prefixed length of tuples; however, the methodology can be applied to most of the drift detection methods present in the MOA framework. In

```

Let  $\alpha$  be the maximum number of base classification algorithms used.
Let  $l$  be the number of base classification selected (with  $l < \alpha$ ).
Let  $M$  be the maximum number of classifiers composing the ensemble.
Let wind the size of the window examined.
Given  $T_1, T_2, \dots, T_{\infty}$ , the infinite tuples composing the stream, analyze in windows of size  $n$ ,
named  $\{T_{w_1}, T_{w_2}, \dots, T_{w_{\infty}}\}$ .
where  $T_i = \{T_{i_1}, T_{i_2}, \dots, T_{i_m}\}$  is a tuple with associated a number of attributes  $m$  and a class.
The current ensemble  $E = \{C_1, C_2, \dots, C_M\}$ 
for each ( $T_{w_j}$ )
  if (drift_detection_function ( $T_{w_j}$ ))
    Partition the tuples composing  $T_{w_j}$  in a training set and a validation set: Traini and Validi
    Train  $\alpha$  different classification algorithms on Traini
    Select the  $l$  classifiers obtaining the best accuracy on the training set.
    Add these classifiers to the ensemble  $E$ 
    if ( $|E| > M$ )
      prune the ensemble  $E$ , removing  $M - |E|$  classifiers, by using a strategy of pruning.
    end if
    Build  $l$  decision profile matrixes, one for each of the classifiers,  $DP_1, DP_2, \dots, DP_l$  using the validation set,
    one for each classifier of dimension  $k \times c$ , where  $k$  is the number of tuples and  $c$  the number of classes.
    Run the distributed GP tool on the validation set Validi in order to obtain
    the combiner function of the ensemble.
    Obtain an Ensemble  $E$ , a combiner function  $FC$ ,
  else
    Build the decision profile matrix  $DP$  of the entire ensemble  $E$ ,
    where each element  $H_{i,j}(x)$  is the support that classifier  $h_i$  gives to the hypothesis that
    the tuple  $x$  comes from class  $\omega_j$ .
    Compute for each class  $j$ :  $\mu_j(x) = \sum H_{i,j}(x)$ .
    Compute the class by using the formula  $class(x) = \operatorname{argmax}_j(\mu_j(x))$ .
  end if
end for each

```

**Fig. 5.9.** The pseudo-code of the algorithm with drift detection.

this work there is not a comparison between different strategy of drift but the choice of drift algorithms used in the experiments is based on work [57].

The main steps of the process are described in the following. An infinite stream of tuples is the input of the model generator and they are analyzed in windows of prefixed length  $n$ . On each window, a function verifies whether a drift occurred; only in the positive case, the labelled tuples of the current window are partitioned equally into training and validation set. The training set is used to train the base classifier algorithms selected among the available in the MOA tool and better specified in the experimental section. Among them, the  $l$  classifiers having the better accuracy on the training set are chosen and added to the current ensemble. A replacing strategy removes some classifiers from the ensemble, whether the maximum number of elements is reached. Afterwards, the GP tool is run to generate the combiner function, which will be applied to the ensemble, by using the validation set.

Finally, if no drift is detected, the new incoming tuples are classified using the schema illustrated before: for each tuple  $x$ , for each possible class  $j$ , the supports are computed using the formula:

$$\mu_j(x) = \sum H_{i,j}(x)$$

and the final classification is computed using the formula



$$class(x) = \operatorname{argmax}_j(\mu_j(x))$$

### 5.5.1 Drift detection and replacement strategies

This Section describes the drift detection algorithms used to handle changes in the data and the replacement strategies used to update the base classifiers of the ensemble. No new drift detection strategy is developed, but the algorithms included in the MOA tool are integrated in the main architecture. In particular, on the basis of the experiments reported in [57], which analyze the performance of the most important drift detection strategies, the STEPD and ADWIN methods are used as base algorithm of the module that detect drifts.

Statistical Test of Equal Proportions (STEPD) [73] is a drift detection algorithm based on the accuracy. It computes two statistics: the overall accuracy from the beginning of the stream and the accuracy of the model computed on a testing window  $W$ . If the difference between the accuracy computed on  $W$  and the overall accuracy is greater than a threshold, then a drift is detected and the model must be rebuilt/updated. In the implementation described in the above-cited paper, the instances stored in memory are used to update the classifier when a drift is detected. The ensemble algorithm updates the models by using the instances of the last window. The window size is fixed, and it contains the target instance (when STEPD detects drift) and its neighbors. The threshold value  $P$  is computed as the percentile of the standard normal distribution in order to obtain the observed significance level. STEPD uses three parameters: the window value to detect recent changes (by default the value is fixed to 20 instances, as in original paper) and the significance levels  $\alpha_w$  and  $\alpha_d$ . In practice, STEPD stores the instances in its memory when  $P < \alpha_w$  and it resets all the variables (i.e., clear its memory, reset the window accuracy, etc.) when  $P < \alpha_d$ . As suggested in the original paper, the default values used are 0.03 for  $\alpha_d$  and 0.08 for  $\alpha_w$ .

The ADaptive WIndowing method [12] keeps a sliding window  $W$  with the most recent examples and compares the distribution on two sub-windows of  $W$ . When the difference of the average value of the two sub-windows is greater than a threshold, then the older sub-window is dropped and a change in the distribution of examples is assigned. However, this idea is computationally intensive and requires a huge amount of memory. Therefore, the current implementation is based on a different data structure. The input records are stored in exponential histograms, a data structure that maintains an approximation of the number of  $I$  contained in a sliding window of length  $W$  using logarithmic memory and update time. The length  $W$  is updated to fit stream variations. The input of the algorithm are real numbers in the interval  $[0,1]$  and to detect drift, only values of 0 and 1 are processed: wrong predictions are marked as 1, good one as 0. A confidence parameter  $\delta$  with value of 0.002 is used to control the false positive rate, i.e. if the expected value of distribution remains constant within  $W$ , the probability that ADWIN shrinks the window at this step is at most  $\delta$ .

As for the strategies for replacing the classifiers composing the ensemble with the new generated by the algorithm, three strategies are selected, named old, best and wheel selection. In the training phase, the dataset is partitioned into two parts: 50% of the instances are selected to train classifiers and the remaining instances compose a validation set to build the ensemble model. The old strategy replaces 1/3 of classifiers in the ensemble with the new generated. The classifiers are removed considering the insertion time in the ensemble model, i.e. the oldest are removed. The new classifier is chosen by considering the accuracy result on the validation set. The best strategy works as the previous one, but it removes classifiers having the worst accuracy on the validation set and it inserts the best ones. As in the previous case, only a 1/3 of classifiers are replaced. Finally, the wheel selection strategy is a selection algorithm based on the wheel mechanism. When a drift is detected, all the classifiers in the repository are trained on the training dataset. Then, classifiers are replaced with a roulette wheel selection algorithm. The performance on validation set is used as a probability, then 1/3 classifiers are selected with probability  $p_i$  using formula  $p_i = \frac{f_i}{\sum_{j=1}^N f_j}$  where  $f_i$  is the accuracy on the validation set.

## Experimental results

In this Chapter, the datasets used in the experiments and the performance of the framework are shown. The main goals are quantifying the performance of the approach with different use cases and understanding the limits and the advantages comparing to existing algorithms. The strategies presented in the previous chapters are analyzed separately and compared with similar approaches.

The results show how the approach can handle unbalanced data and missing features and the software is tested with different dataset to detect advantages and disadvantages. The experiments were conducted on both artificial and real datasets to evaluate the performance on targeted tests (e.g. different values of missing data are created artificially) and in real environment.

Furthermore, a set of experiments are conducted to test the ability handling changes. The first set uses specialized models, then the results using configuration with the drift detection algorithms are presented. The well-known metrics like error/accuracy, precision, recall, AUC are used to evaluate the results.

### 6.1 Description of the datasets used in the experiments

Many datasets are used in the experiments to test the different problems of missing data, unbalanced data, intrusion detection, etc. Some datasets can be used in multiple cases; others can be used to test only a particular case. As example, a dataset with many features and unbalanced classes can be used to test the capacity of handling unbalanced data but also in the case of missing data because the dataset can be partitioned in multiple subset grouping the attributes. For this reason, a general description is presented in this Section and the specific details are also provided when the results of the experiments are discussed.

#### Unbalanced datasets

The following datasets are selected to test the capacity of the algorithm handling unbalanced data. Each dataset has one or more classes with a very low number of

instances compared with other classes. Moreover, these datasets are very common in the machine learning literature about classification.

The main characteristics of each dataset are showed in Table 6.1: the size, the number of features and of classes and the percentage of the minority class of the datasets used in the experiments. The datasets present different characteristics in terms of number of attributes and classes; in addition, most of them have a distribution of the tuples belonging to one or more classes really unbalanced, as it is evident from the percentage of the minority class. The DNA dataset come from the UCI KDD Archive<sup>1</sup>, the Pendigit and the Satimage are taken from the UCI Machine Learning Repository<sup>2</sup>, the Phoneme dataset is from the ELENA project<sup>3</sup>.

The DNA dataset is composed by splice junctions which are points on a DNA sequence at which ‘superfluous’ DNA is removed during the process of protein creation in higher organisms. The problem posed in this dataset is to recognize, given a sequence of DNA, the boundaries between exons (the parts of the DNA sequence retained after splicing) and introns (the parts of the DNA sequence that are spliced out).

The Pendigit dataset is designed for the pen-based recognition of handwritten digits and is created by collecting 250 samples from 44 writers. Each sample is the handwritten numerals 0-9 and the number is the class.

The Satimage dataset was generated taking a small section (82 rows and 100 columns) from the original data, the landsat dataset. It consists of the multi-spectral values of pixels in 3x3 neighbourhoods in a satellite image, and the classification associated with the central pixel in each neighbourhood. The aim is to predict this classification, given the multi-spectral values. In the sample database, the class of a pixel is coded as a number.

For the Phoneme dataset, the aim is to distinguish between nasal (class 0) and oral sounds (class 1). The class distribution is 3,818 samples in class 0 and 1,586 samples in class 1.

**Table 6.1.** Description of datasets ordered by decreasing percentage of minority class.

Dataset	Number of examples	Number of features	Number of Class	Minority Class
Satimage	6,435	36	6	0.0972
DNA/Splice	3,190	61	3	0.2404
Phoneme	5,404	5	2	0.2938
Pendigit	10,992	16	10	0.0959
KDDCup	494,020	41	5	1.052E-4

<sup>1</sup> <http://kdd.ics.uci.edu>.

<sup>2</sup> <http://www.ics.uci.edu/mlearn/MLRepository.html>

<sup>3</sup> <ftp://dice.ucl.ac.be> in the directory `pub/neural/ELENA/databases`.

### Datasets for the missing data experiments

In Table 6.2, the characteristics of three datasets are illustrated, with a significant number of features and they are used for the comparison of the capacity of the framework in handling missing data with the work in [80]. The dataset are Covtype, M-Feat, OCR and DNA datasets, Pendigit and Satimage.

The Covtype dataset is a real large dataset, representing the prediction of forest cover type from cartographic variables determined by the U.S. Forest Service and the U.S. Geological Survey. The task of classifying this dataset is not easy, especially in the presence of missing attributes, as it contains 44 binary attributes out of 54 totals, representing qualitative independent variables such as wilderness areas and soil type.

The M-Feat dataset consists of features of handwritten numerals (0-9) extracted from a collection of Dutch utility maps. 200 patterns per class (for a total of 2,000 patterns) have been digitized in binary images. These digits are represented in terms of the following six feature sets:

1. mfeat-fou: 76 Fourier coefficients of the character shapes;
2. mfeat-fac: 216 profile correlations;
3. mfeat-kar: 64 Karhunen-Love coefficients;
4. mfeat-pix: 240 pixel averages in 2 x 3 windows;
5. mfeat-zer: 47 Zernike moments;
6. mfeat-mor: 6 morphological features.

The OCR dataset is the result of an extraction of normalized bitmaps of handwritten digits from a preprinted form. Each bitmap, sized 32x32, is divided into nonoverlapping blocks of 4x4 and the number of on pixels are counted in each block. This generates an input matrix of 8x8 where each element is an integer in the range 0..16. This reduces dimensionality and gives invariance to small distortions. The total number of attributes is 64.

**Table 6.2.** Description of the additional datasets used for the missing data experiments.

Dataset	Number of examples	Number of features	Number of Class	Partitions
OCR	5,620	62	10	3
M-Feat	2,000	216	10	4
Covtype	581,012	54	7	3

### The Unix dataset

The Unix Users Data is created by Greenberg [49]. It contains the commands typed in a Unix shell by 168 users with different level of skills. Each user is assigned to one of four profiles: nonprogrammer, novice programmers, experienced programmers and computer scientist.

The novice programmers profile represents users with little exposure to programming and command-based interfaces. The experienced programmers members have fair knowledge with the unix environment. The computer scientist group contains users with an extensive knowledge in programming and command-line tools. The latter profile consists of users with no programming skills, their dominant activities are the processing and preparation of documents. This dataset is preprocessed in the same way used in [54]. For each user the first 100 and 500 commands typed are considered; then the commands subsequences of fixed length (from 3 to 6) are extracted from the list. Each user represents a record in the processed dataset and all the distinct subsequences are used as record attribute and the attribute value is the number of times the subsequence is typed by the user (0 means that the subsequence is never typed).

### KDDCup'99

KDDCup 1999<sup>4</sup> is a well-known dataset in the cyber security domain. This dataset contains 494,020 records, representing normal connections and 24 different attack types. Each attack is clustered into four main categories, so each connection belongs to the following classes: normal (normal, i.e., no attack), DoS (Denial of Service connections), R2L (Remote to User, remote attacks addressed to gain local access), U2R (User to Root, exploits used to gain root access) or Probe (probing attack to discover known vulnerabilities).

### The ISCX dataset

An improvement of the KDD99 dataset is ISCX IDS, consisting of 2,230,620 records, fully labelled, representing realistic traffic scenarios of 7 days, containing different types of attack, i.e., HTTP Denial of Service, DDos, Brute Force SSH and attempts of infiltrating the subnetwork from the inside (see Table 6.3).

Day	Description	Size of the pcap file (GB)	Number of Flows	Percentage of Attacks
Day 1	<i>Normal traffic without malicious activities</i>	16.1	359,673	0.000 %
Day 2	<i>Normal traffic with some malicious activities</i>	4.22	134,752	1.545 %
Day 3	<i>Infiltrating the network from the inside &amp; Normal traffic</i>	3.95	153,409	6.395 %
Day 4	<i>HTTP Denial of Service &amp; Normal traffic</i>	6.85	178,825	1.855 %
Day 5	<i>Distributed Denial of Service using an IRC Botnet</i>	23.4	554,659	6.686 %
Day 6	<i>Normal traffic without malicious activities</i>	17.6	505,057	0.000 %
Day 7	<i>Brute Force SSH + Normal activities</i>	12.3	344,245	1.435 %

**Table 6.3.** Main characteristics of the ISCX IDS dataset.

DARPA<sup>5</sup> and KDD99 are two very popular datasets used in the classification in the IDS domain, see Travallae et al. [91]; however, they have been thoroughly criticized for being unable to provide a realistic scenario. To overcome this issue, this set of experiments are based on the ISCX IDS dataset from the Information Security

<sup>4</sup> <http://www.sigkdd.org/kdd-cup-1999-computer-network-intrusion-detection>

<sup>5</sup> <https://www.ll.mit.edu/ideval/data>

Centre of Excellence of the University of New Brunswick [85]. The ISCX dataset is the result of capturing seven days of network traffic in a controlled testbed made of a subnetwork placed behind a firewall. Normal traffic was generated with the aid of agents that simulated normal requests of human users following some probability distributions extrapolated from real traffic. Attacks were generated with the aid of human operators. The result is a fully labelled dataset containing realistic traffic scenarios. Indeed, the dataset consists of standard pcap (packet capture) files one for each day containing the relative network traffic, as illustrated in Table 6.3. Different days contain different attack scenarios, ranging from HTTP Denial of Service, DDos, Brute Force SSH and attempts of infiltrating the subnetwork from the inside. The main characteristics of this dataset are summarized in Table 6.3.

### Artificial datasets

The performances of the framework with streaming data are evaluated on an artificial dataset and on a real dataset of the cyber security domain. To generate the artificial dataset, the HyperPlane generator available in MOA<sup>6</sup> is used. This generator is very popular as benchmark for drift detection algorithms. It was originally used in (Hulten et al. 2001) [53]. It generates data for a binary classification problem, taking a random hyperplane in  $d$ -dimensional Euclidean space as a decision boundary. The user can customize the number of attributes generated, the attributes used to generate drift (the others can be considered as irrelevant), the magnitude of changes and the percentage of noise to add to the data.

## 6.2 Experimental section on unbalanced data

In this section, in order to assess the goodness of the proposed framework handling unbalanced data, using the parameters and the datasets described in the next subsection, an analysis of the model size and the accuracy obtained by CAGE-Combiner is presented, with different configurations and compared the approach with different state-of-the-art combination strategies (subsection 6.2.2). Then, the performance of the approach was analyzed on a really unbalanced and hard intrusion detection dataset (subsection 6.4.1).

### 6.2.1 Environment configuration and parameter settings

All the experiments were performed on a Linux cluster with 16 Itanium2 1.4GHz nodes, each having 2 GBytes of main memory and connected by a Myrinet high performance network. No tuning phase has been conducted for the GP algorithm, but the same parameters used in the original paper [36] were used, listed in the following: a probability of crossover equal to 0.7 and of mutation equal to 0.1, a maximum depth equal to 7, and a population of 132 individuals per node. The algorithm was run on

<sup>6</sup> <http://moa.cms.waikato.ac.nz>

4 nodes, using 1000 generations and the original training set was partitioned among the 4 nodes. The parsimony factor is varied using the values of 0, 0.01 and 0.1 in order to generate classifiers of different size and to study the effect of the size of classifiers on the classification error and on the generalization of the algorithm. All the results were obtained by averaging 30 runs.

The process of model building is based on three partitions of the original dataset.

The different classifiers composing the ensemble are trained on the same training set. In practice, each dataset is partitioned in three subsamples: the 70% of original dataset is used to train the base classifiers, the remaining 30% is equally partitioned in two parts: validation and test set. The validation part is used by the evolutionary algorithm to build the combination function, while the error rate of the best tree is calculated on the test partition. The learning algorithms are implemented in WEKA platform and the models are built using standard parameters.

The algorithms used as base classifiers in the experiments are based on the WEKA implementation<sup>7</sup> and are listed in the following: J48 (decision trees), JRIP rule learner (Ripper rule learning algorithm), NBTree (Naive Bayes tree), Naive Bayes, 1R classifier, logistic model trees, logistic regression, decision stumps and 1BK (k-nearest neighbor algorithm).

**Table 6.4.** Error rate of the base classifiers used to build the ensemble

Dataset	Type	J48	Jrip	NBTree	NaiveBayes	OneR	LMT	Logistic	DecisionStump	Ibk
Satimage	training	2.60	7.90	17.30	19.90	39.10	8.80	12.00	55.80	0.00
	validation	15.00	13.20	19.10	20.40	43.40	13.20	14.50	58.30	9.30
	test	15.30	14.20	20.60	22.90	41.30	13.60	15.80	56.60	10.10
Phoneme	training	8.40	11.30	10.80	23.10	18.20	9.00	24.60	24.30	0.00
	validation	14.40	14.90	13.90	22.90	25.60	12.80	25.40	24.80	9.40
	test	14.80	16.60	14.30	25.20	23.20	15.20	26.00	25.40	11.30
Pendigit	training	0.70	1.20	0.20	13.50	59.40	0.30	3.70	79.70	0.00
	validation	4.10	4.50	4.90	14.90	63.50	1.70	4.40	79.20	1.00
	test	3.90	3.30	5.20	14.90	60.50	1.90	4.80	79.80	0.50
Dna	training	4.00	4.20	0.00	3.30	0.00	0.00	0.00	37.00	0.00
	validation	5.50	4.80	8.30	5.00	71.40	3.50	11.30	40.90	27.80
	test	4.50	4.30	5.50	4.30	73.70	3.80	11.00	37.80	24.10

In Table 6.4, it is shown the error rate of each base classifier respectively on the training, validation and test set and this helps to understand the improvement obtained in terms of accuracy using an ensemble, as shown in the next subsection.

### 6.2.2 Comparing with other evolutionary strategies and ensemble techniques

As stated in the previous subsection, the GP framework is executed without any tuning of the parameters. The only exception is the analysis (Table 6.5) of the effect of the combiner function size on the accuracy, i.e. the ratio of the number of correctly

<sup>7</sup> <http://www.cs.waikato.ac.nz/ml/weka>



classified tuples to the total number of tuples, varying the value of the parsimony factor.

Generally, when using GP-based algorithms, there are different methods to limit the uncontrolled growth of the average size of an individual in the population (bloat problem); the simplest way is to limit their maximum depth and to punish individuals of excessive size. In the depth analysis presented in [69], the effect of many other complex methods are experimentally tested, but none of them results predominant over the other to justify the complexity introduced. Therefore, the widely used method of parsimony is adopted, consisting in simply adding to the fitness function a penalty given by the product of a constant parameter (the parsimony factor) and of the overall number of nodes and leaves of the genetic programming tree. Typically, the higher the parsimony, the simpler the tree, but the accuracy could diminish. The parsimony factor is varied using the values of 0 (no parsimony), 0.01 and 0.1.

**Table 6.5.** The error rate for different values of parsimony (0, 0.1 and 0.01), along with the average number of classifiers and functions used in the best tree.

Dataset	Parsimony	Error Train	Error Test	Distinct Classifiers	Total Classifiers	Functions
Satimage	0	<b>7.77</b> $\pm$ 0.60	<b>9.08</b> $\pm$ 0.56	8.64 $\pm$ 0.79	78.44 $\pm$ 42.03	30.56 $\pm$ 15.01
	0.01	<b>7.46</b> $\pm$ 0.62	<b>9.25</b> $\pm$ 0.58	7.26 $\pm$ 1.34	25.70 $\pm$ 11.49	11.10 $\pm$ 4.16
	0.1	<b>7.48</b> $\pm$ 0.41	<b>9.09</b> $\pm$ 0.51	6.57 $\pm$ 1.54	14.46 $\pm$ 4.61	6.76 $\pm$ 2.45
Phoneme	0	<b>8.27</b> $\pm$ 0.43	<b>11.63</b> $\pm$ 1.30	8.70 $\pm$ 0.55	99.95 $\pm$ 74.63	38.85 $\pm$ 30.36
	0.01	<b>7.62</b> $\pm$ 0.65	<b>11.14</b> $\pm$ 0.44	6.61 $\pm$ 1.41	26.15 $\pm$ 18.37	11.96 $\pm$ 6.98
	0.1	<b>7.80</b> $\pm$ 0.46	<b>10.91</b> $\pm$ 0.51	5.53 $\pm$ 1.33	13.73 $\pm$ 5.47	7.00 $\pm$ 2.75
Pendigits	0	<b>0.66</b> $\pm$ 0.22	0.74 $\pm$ 0.22	8.86 $\pm$ 0.33	71.30 $\pm$ 37.16	27.95 $\pm$ 14.82
	0.01	<b>0.60</b> $\pm$ 0.12	0.68 $\pm$ 0.12	6.13 $\pm$ 1.50	14.48 $\pm$ 7.84	6.10 $\pm$ 3.30
	0.1	<b>0.64</b> $\pm$ 0.10	0.67 $\pm$ 0.12	6.13 $\pm$ 1.08	10.40 $\pm$ 3.20	5.16 $\pm$ 2.35
Dna	0	<b>2.46</b> $\pm$ 0.85	3.71 $\pm$ 1.05	8.48 $\pm$ 0.89	88.31 $\pm$ 92.59	33.86 $\pm$ 36.10
	0.01	<b>1.86</b> $\pm$ 0.15	3.48 $\pm$ 0.28	6.53 $\pm$ 0.92	11.70 $\pm$ 2.53	4.50 $\pm$ 1.25
	0.1	<b>1.82</b> $\pm$ 0.13	3.53 $\pm$ 0.22	6.26 $\pm$ 0.81	9.20 $\pm$ 1.75	4.30 $\pm$ 1.29

For each dataset, the experiments are conducted to show different values of the parsimony factor and the highlighted values are the ones having statistically significant differences using the Friedman test. The critical value of the Friedman test [32] is obtained from a chi-square distribution with two degree of freedom and the number of cases considered is 30 for each set. A significancy level of 5% is used.

In all the tables in this Section, the following rules are used: the parsimony value is underlined when the Friedman test, comparing experiments with same value of parsimony but different values of the missing data, presents statistically significant differences. The accuracy/error values are marked in bold when the Friedman test, comparing experiments with the same missing percentage but with different parsimony factors, presents statistically significant differences.

In Table 6.5, as only the behavior of the algorithm when the parsimony factor is changed is considered, values in bold represent significantly different results in terms of parsimony. In two of the four datasets, the differences in terms of error rates are significant statistically; however, the differences are not remarkable. On the contrary,

the size of the trees and the distinct classifiers selected by the algorithm are greatly affected by the parsimony factor. For this reason, a parsimony factor of 0.1 for the other experiments conducted on the following is used.

**Table 6.6.** Error rate for different strategies for the 4 datasets used in the experiments.

	Satimage	Phoneme	Pendigit	DNA
CAGE-Combiner	9.09	10.92	0.68	3.53
EVEN	8.91	11.68	0.68	4.20
EVEN (cut-off = 0.8)	8.69	11.06	0.66	4.34
Majority Vote	10.52	15.85	0.98	4.20
Weighted Vote	10.40	15.04	0.93	4.32
Best classifier	10.60	12.59	0.89	4.82
Stacking NB	10.75	14.93	0.81	4.55
Stacking LR	9.72	11.12	0.82	5.03

In table 6.6, CAGE-Combiner is compared with the EVEN algorithm [90] and also with the meta-algorithms used in the same paper. Note that EVEN uses a population size of 120 (the number of classifiers) for 1000 generations. The results show that CAGE-Combiner obtain better or comparable accuracy for all the datasets; however, the number of classifiers used is sensibly minor than the 120 used by the EVEN algorithm. In the latter, a cut-off threshold is introduced and only those classifiers whose weights are above this threshold are allowed to participate to the ensemble. The maximum value of cut-off used in the paper (0.8) and shown in the table permits to reduce the number of classifiers to about 25% of the original size, while CAGE-Combiner (see Table 6.5) using the parsimony value of 0.1, obtains a better reduction of the number of classifiers (about 10%), without any relevant reduction in term of accuracy.

### 6.2.3 Results for the KDDCup'99 dataset

To evaluate the system proposed on a real-world dataset in the field of cyber security, the same experiments as the previous subsection are performed using one of the most used dataset for the task of classification of intrusions: the KDDCup'99 dataset.

**Table 6.7.** The error rate for different values of parsimony (0, 0.1 and 0.01), along with the average number of classifiers and functions used in the best tree: KDD Cup 99.

Pars.	Error	Distinct Cls	Total Cls	Functions	DoS	Normal	Probe	R2L	U2R
0	0.0106 ± 0.0015	7.60 ± 0.71	65.23 ± 50.46	26.53 ± 19.03	0.0000	0.0003	0.0114	0.0506	0.2000
0.01	0.0105 ± 0.0012	6.20 ± 1.01	13.30 ± 6.76	5.80 ± 2.65	0.0000	0.0003	0.0109	0.0510	0.2333
0.1	0.0121 ± 0.0016	5.37 ± 0.80	9.03 ± 3.01	3.80 ± 1.45	0.0000	0.0003	0.0106	0.0490	0.2667

In Table 6.7, it is evident that the size of the trees and the distinct classifiers selected by the algorithm strongly depends on the parsimony factor, while for the accuracy the differences are minimal and the best results are obtained with the parsimony value of 0.01.

Furthermore, the goal of this experiment is to understand the behavior of the framework for the unbalanced datasets and in particular, for the minority classes of the KDD Cup dataset, i.e., Probe, R2L and U2R.

To this aim, the work in [5] is considered as competitor, which describes a boosting approach, named Greedy-Boost, to build an ensemble of classifiers based on a linear combination of models, specifically designed to operate for the intrusion detection domain. The main idea of the Greedy-Boost algorithm is to extend the boosting process maintaining the models that behave better on the examples badly predicted in the previous round of the boosting algorithm (while the classical algorithm adjust only the weights and not the models).

In Table 6.8, CAGE-Combiner is compared with the Greedy-Boost algorithm on the KDDCup 99 datasets and the precision and the recall values are reported for all the classes. It is evident how the CAGE-Combiner performs better both for the precision and the recall measure, especially in the case of the minority classes R2L and U2R.

**Table 6.8.** Precision and Recall for different strategies for the KDD Cup dataset. In the first column, it is reported the class distribution for the test set.

	Class Distribution	Precision		Recall	
		Greedy-Boost	CAGE-Combiner	Greedy-Boost	CAGE-Combiner
DoS	0.7960	100.0	100.0	100.0	100.0
Normal	0.1936	99.1	99.9	100.0	100.0
Probe	0.0079	99.0	99.6	97.1	98.9
R2L	0.0023	93.2	98.5	71.9	94.9
U2R	4.85E-5	88.5	93.1	44.2	76.7

### 6.3 Experimental results on missing features

In this section, the experiments on handling missing features are described together with the main parameters. In addition to a number of well-known benchmark datasets, a real and hard dataset was used to validate the approach: the Unix dataset. Finally, the framework is compared with a correlated work for the case of missing data.

#### 6.3.1 Parameter settings

All the experiments were performed on a Linux cluster with same characteristics described in the previous section.

Each dataset is partitioned into three subsamples: 70% of original dataset is used to train the classifiers, which will compose the ensemble, the remaining 30% is equally partitioned into two parts: validation and test set. The validation part is used by the evolutionary algorithm to build the combination function of the ensemble, while the error rate, i.e. the ratio of the number of misclassified tuples to the total number of tuples, of the best tree is calculated on the test partition. The learning algorithms are implemented in the WEKA tool and the different models are built by using standard parameters.

More in detail, the algorithms used as classifiers in the experiments are based on the WEKA implementation<sup>8</sup> and are listed in the following: J48 (decision trees), K random tree, Function Tree (based on logistic regression) JRIP rule learner (Ripper rule learning algorithm), ConjunctiveRule, NBTree (Naive Bayes tree), Naive Bayes, DTNB (decision table with naive bayes), 1R classifier, logistic model trees, logistic regression, decision stumps and 1BK (k-nearest neighbor algorithm).

The results of these classifiers are showed in Table 6.4.

#### 6.3.2 Experiments on the capacity of handling missing features

Two sets of experiments were performed in order to evaluate the ability of the CAGE-MetaCombiner approach in handling missing features. The first set analyzes the behavior of the algorithm in respect of different percentage of tuples with missing data and different values of the parsimony factor affecting the size of the solutions. The second set compares the approach with a related work: the Learn++.MF framework [80], described in the related work section.

For these experiments, in addition to the OCR, M-Feat and Pendigits datasets, used by the Learn++.MF framework, the Satimage, Dna and Covtype datasets are also used, which present a significant number of features. It is worth remembering that the main goal of this section is understanding the case in which entire groups of features are missing and not in coping with random patterns of missing features. Therefore, as the features of the above-mentioned datasets are logically divided into

---

<sup>8</sup> <http://www.cs.waikato.ac.nz/ml/weka>

groups, the datasets OCR, M-Feat, Satimage, Pendigits, Dna and Covtype are partitioned respectively in 3, 4, 3, 2, 3 and 3 partitions, trying to not separate correlated (or coming from the same source) attributes. Then, to simulate the missing data, for each partition a tuple can be removed according to a probability threshold, i.e., this parameter controls the percentage of tuples, which have missing attributes. For instance, if this parameter is set to 10%, the entire partition of the features belonging to this tuple has a probability of 0.1 to be missing. If all the partitions of a tuple are missing, this tuple will be considered as an error of classification. The values of the threshold are in the range 0-40%, with an interval of 10%, with 0% means no missing data.

**Table 6.9.** The error rate of CAGE-MetaCombiner using three parsimony values and different percentages of missing data.

Dataset	Pars.	Missing Percentage					Classifiers		
		0%	10%	20%	30%	40%	Distinct	Total	Functions
OCR	0	5.43 ± 0.26	6.37 ± 0.21	7.90 ± 0.35	10.27 ± 0.37	14.16 ± 0.57	8.77 ± 0.12	84.80 ± 4.36	33.97 ± 0.21
	0.1	5.36 ± 0.27	6.19 ± 0.28	7.65 ± 0.40	10.31 ± 0.39	13.99 ± 0.62	7.50 ± 0.22	21.20 ± 0.93	9.10 ± 0.64
	0.4	5.30 ± 0.22	6.23 ± 0.20	7.54 ± 0.39	9.95 ± 0.35	13.85 ± 0.41	6.33 ± 0.19	11.63 ± 0.24	5.23 ± 0.41
Pendigits	0	4.96 ± 0.28	6.81 ± 0.34	10.21 ± 0.36	15.09 ± 0.29	21.31 ± 0.35	8.95 ± 0.05	104.75 ± 13.75	41.95 ± 3.25
	0.1	4.97 ± 0.27	6.88 ± 0.19	10.12 ± 0.42	14.89 ± 0.19	21.35 ± 0.55	7.75 ± 0.35	25.55 ± 1.05	11.30 ± 0.30
	0.4	5.20 ± 0.34	7.07 ± 0.40	10.49 ± 0.48	15.21 ± 0.50	21.24 ± 0.35	5.70 ± 0.60	11.05 ± 2.95	4.95 ± 1.25
Dna	0	7.33 ± 1.40	9.29 ± 1.45	11.91 ± 1.26	15.02 ± 0.94	18.98 ± 0.73	8.77 ± 0.19	85.33 ± 6.36	33.67 ± 2.72
	0.1	7.86 ± 1.52	9.93 ± 1.16	12.55 ± 1.32	16.11 ± 1.30	20.10 ± 0.57	6.29 ± 1.23	18.17 ± 5.69	8.46 ± 1.94
	0.4	7.31 ± 0.77	9.49 ± 0.65	12.05 ± 0.90	15.56 ± 0.83	19.21 ± 1.33	5.77 ± 0.38	9.47 ± 1.16	4.13 ± 0.80
M-Feat	0	4.85 ± 0.22	4.95 ± 0.24	5.35 ± 0.30	6.11 ± 0.44	7.70 ± 0.51	8.10 ± 0.12	53.80 ± 14.24	21.23 ± 5.58
	0.1	4.93 ± 0.22	5.04 ± 0.18	5.32 ± 0.25	6.17 ± 0.41	7.66 ± 0.75	5.36 ± 0.29	9.94 ± 0.88	4.53 ± 0.85
	0.4	4.78 ± 0.14	4.97 ± 0.20	5.20 ± 0.17	5.97 ± 0.28	7.64 ± 0.58	3.20 ± 1.27	4.28 ± 2.00	1.77 ± 1.16
Satimage	0	12.62 ± 0.23	12.84 ± 0.26	13.54 ± 0.28	15.05 ± 0.34	17.13 ± 0.40	8.80 ± 0.14	74.60 ± 13.45	30.53 ± 5.33
	0.1	12.63 ± 0.17	12.86 ± 0.12	13.59 ± 0.20	15.02 ± 0.24	17.79 ± 0.31	7.07 ± 0.69	18.23 ± 1.24	8.47 ± 0.59
	0.4	12.62 ± 0.20	12.86 ± 0.21	13.64 ± 0.20	14.84 ± 0.41	17.53 ± 0.56	5.30 ± 0.57	8.73 ± 0.73	3.40 ± 0.42
Covtype	0	21.95 ± 1.09	23.08 ± 0.89	24.54 ± 0.72	26.43 ± 0.54	28.95 ± 0.43	8.90 ± 0.08	123.87 ± 26.44	50.43 ± 9.73
	0.1	21.81 ± 0.83	22.69 ± 0.50	24.22 ± 0.40	26.25 ± 0.32	28.80 ± 0.27	8.33 ± 0.12	57.00 ± 4.65	26.07 ± 2.04
	0.4	21.71 ± 0.54	22.90 ± 0.45	24.38 ± 0.41	26.32 ± 0.33	28.89 ± 0.27	7.20 ± 0.36	32.27 ± 11.01	14.80 ± 4.81

Table 6.9 shows the error of classification of the CAGE-MetaCombiner algorithm by varying the parsimony factor using the values of 0 (no parsimony), 0.1 and 0.4, using the above-defined percentages of missing features. Therefore, the effect of the two parameters are evaluated: the parsimony factor, to reduce the complexity of the overall meta-ensemble, and the percentage of missing values, to evaluate the capacity of the algorithm to handle missing data. The Friedman test is computed in the same way specified in the previous subsection to highlight statistically significant differences. Varying the parsimony, for each value of missing data, there is no significant difference in most of the cases. On the contrary, by varying the percentage of missing data, the differences for the error rate are statistically significant with the exception of the Covtype dataset for the cases of 30% and 40% of missing data.

For all the datasets, and for a percentage of missing features up to 20%, the degradation in accuracy is moderate (it is always less than 3%), while using 30% or 40% as values of the threshold, the error has a remarkable increase (in some cases, it arrives at 7%). More specifically, the error for the M-Feat and for the Satimage dataset does not deteriorate much, even though a threshold of 30% and 40% is used.

As for the Covtype dataset, the number of distinct classifiers selected by the algorithm does not vary much (from 7 to 9), while the average size of the tree is substantially different; therefore, with a parsimony factor set to 0.4, the algorithm performs quite well and the size of the overall ensemble is compact. Increasing the missing percentage threshold from 0% to 40%, the degradation of the accuracy is limited. These results confirm the effectiveness of CAGE-MetaCombiner in handling missing data.

The comparison between Learn++.MF and CAGE-MetaCombiner is evaluated on the three datasets, which are also used in [80] and the results are shown in Table 6.10. For CAGE-MetaCombiner, the value of 0.4 is used for the parsimony factor. For Learn++.MF, the number of missing features is chosen to be equal to the size of the partition used by our algorithm.

Owing to the strategy of computing the base classifiers used by Learn++.MF algorithm, the ability to handle missing data is determined by how many features are missing. So the missing percentage values, reported in the original paper, arrive at 30%, and in almost all cases both the algorithms are able to classify each dataset with a reasonable accuracy. As for the M-Feat dataset, the 'n/a' value in the table means that no classifiers are usable for instances with more of 30% of missing attributes for the Learn++.MF algorithm. This limits the applicability of the algorithm for large percentages of missing data. Anyway, for all the datasets, CAGE-MetaCombiner has better results than Learn++.MF when missing data increase, and this is more evident when the datasets have many features.

**Table 6.10.** Comparison between CAGE-MetaCombiner and Learn++.MF. *nof* represents the number of missing features for each record of the dataset (error rate).

		Percentage of Missing					
		10%		20%		30%	
		CageMC	Learn++.MF	CageMC	Learn++.MF	CageMC	Learn++.MF
	nof						
OCR	20	6.23 ± 0.20	3.50 ± 0.10	7.54 ± 0.39	8.20 ± 0.4	9.95 ± 0.35	13.50 ± 1.10
Pendigits <sup>9</sup>	8	7.07 ± 0.40	10	10.49 ± 0.48	13	15.21 ± 0.50	17
M-Feat	50	4.97 ± 0.20	6.62 ± 0.08	5.20 ± 0.17	7.44 ± 2.87	5.97 ± 0.28	n/a

### 6.3.3 Analysis on a real-world dataset: the Unix dataset.

In this Section the results of the algorithm on a real-world dataset are shown. The description of the Unix dataset is provided in Section 6.1. The goal is to test the approach handling missing data with real data, instead in the previous Section the results are obtained on artificial data.

In Table 6.11, the Cage-MetaCombiner performance on the Unix dataset using different subsequence lengths and different percentage of missing data is shown. The

<sup>9</sup> The value reported concerning the accuracy Learn++.MF for the Pendigit dataset have been taken from a graph of the original paper, in which the values of the standard deviation were not present.

classification rate for the 100 commands experiment is slightly better than for the case with 500 commands probably because the increase in the number of commands could lead to an increment in the number of features and consequently the training phase becomes harder than the previous case. Owing to the high number of features, the results are not much affected by the missing rate, that is the algorithm has good performance with all the percentages of missing data. In all the cases, the results show that the algorithm presents comparable performance by using different values of parsimony.

It's worthwhile to remember the advantage of a high value of parsimony factor. The parameter controls the size of combining functions and the increase of the value is tied to a small tree size (that is both base classifiers and combining operators).

**Table 6.11.** Classification rate (Percent) of Cage-MetaCombiner with the Unix dataset using three parsimony values, different subsequence lengths and with different percentages of missing data.

Commands	Sequence	Parsimony	Percentage of Missing						Distinct Classifiers	Total Classifiers	Functions	
			0%	10%	20%	30%	40%	80%				
100	3	0	83.96 ± 1.90	83.84 ± 3.68	83.70 ± 2.43	82.67 ± 2.91	82.64 ± 2.51	81.32 ± 2.21	7.75 ± 0.45	52.97 ± 9.42	21.62 ± 3.27	
		0.1	85.73 ± 1.22	84.00 ± 1.33	83.60 ± 1.95	83.57 ± 2.07	83.20 ± 2.82	81.69 ± 3.00	4.19 ± 0.31	6.73 ± 0.83	3.51 ± 0.32	
		0.4	84.16 ± 0.35	83.96 ± 1.20	83.90 ± 1.05	83.86 ± 1.64	83.49 ± 1.64	83.00 ± 2.92	3.41 ± 0.28	4.66 ± 0.39	2.49 ± 0.25	
		0	83.81 ± 3.97	83.70 ± 1.89	<b>82.81 ± 1.72</b>	82.54 ± 2.24	<b>81.98 ± 3.65</b>	81.82 ± 2.42	7.71 ± 0.28	44.79 ± 3.04	18.41 ± 1.17	
		0.1	83.86 ± 0.81	83.73 ± 1.24	<b>83.20 ± 1.74</b>	83.10 ± 1.73	<b>82.89 ± 1.51</b>	82.11 ± 3.01	4.50 ± 0.31	7.61 ± 1.51	3.75 ± 0.63	
		0.4	84.76 ± 0.53	84.19 ± 0.54	<b>83.86 ± 0.93</b>	83.76 ± 1.34	<b>83.13 ± 1.70</b>	81.65 ± 2.70	3.57 ± 0.33	4.68 ± 0.38	2.49 ± 0.18	
	4	0	83.37 ± 1.31	<b>82.97 ± 1.54</b>	<b>82.67 ± 2.09</b>	82.00 ± 2.41	81.92 ± 3.25	81.45 ± 2.34	7.52 ± 0.34	43.31 ± 8.23	17.79 ± 3.07	
		0.1	83.96 ± 0.74	<b>83.93 ± 0.81</b>	<b>83.73 ± 1.46</b>	83.33 ± 1.62	83.21 ± 2.02	82.51 ± 2.66	4.57 ± 0.40	7.40 ± 1.17	3.67 ± 0.51	
		0.4	85.14 ± 3.96	<b>83.83 ± 1.15</b>	<b>84.29 ± 1.19</b>	84.09 ± 1.64	83.33 ± 3.63	82.84 ± 2.23	3.77 ± 0.27	4.83 ± 0.58	2.64 ± 0.31	
		0	84.74 ± 3.96	84.09 ± 1.33	83.96 ± 1.89	83.86 ± 1.18	<b>82.05 ± 3.08</b>	81.19 ± 2.24	8.01 ± 0.24	51.23 ± 7.28	20.91 ± 2.51	
		0.1	85.06 ± 0.53	84.23 ± 0.99	84.26 ± 1.29	83.60 ± 1.32	<b>83.09 ± 2.01</b>	82.38 ± 2.43	4.54 ± 0.44	7.61 ± 1.44	3.83 ± 0.63	
		0.4	84.96 ± 0.74	84.19 ± 0.54	83.93 ± 0.55	83.85 ± 1.31	<b>83.64 ± 1.72</b>	82.21 ± 2.32	3.63 ± 0.13	4.77 ± 0.39	2.56 ± 0.16	
	500	3	0	<b>86.70 ± 7.05</b>	<b>85.78 ± 5.83</b>	<b>85.58 ± 5.37</b>	<b>85.08 ± 5.07</b>	<b>85.02 ± 5.27</b>	82.44 ± 3.05	7.87 ± 0.15	50.06 ± 5.56	20.38 ± 2.12
			0.1	<b>84.39 ± 7.36</b>	<b>84.08 ± 7.59</b>	<b>84.26 ± 6.93</b>	<b>83.80 ± 6.84</b>	<b>83.27 ± 5.38</b>	82.01 ± 5.00	4.46 ± 0.65	7.55 ± 1.66	3.93 ± 0.80
			0.4	<b>90.56 ± 2.21</b>	<b>89.58 ± 5.58</b>	<b>88.86 ± 5.40</b>	<b>87.92 ± 5.38</b>	<b>87.46 ± 2.42</b>	82.64 ± 3.51	3.03 ± 0.32	3.77 ± 0.57	2.12 ± 0.30
			0	83.63 ± 1.22	<b>82.78 ± 4.03</b>	82.31 ± 3.44	<b>82.25 ± 3.23</b>	82.10 ± 2.27	82.08 ± 2.28	7.83 ± 0.14	57.41 ± 7.51	22.93 ± 3.28
			0.1	83.86 ± 0.85	<b>83.57 ± 2.15</b>	82.93 ± 1.59	<b>82.10 ± 2.12</b>	81.91 ± 2.70	81.04 ± 3.28	4.21 ± 0.27	6.97 ± 0.84	3.71 ± 0.46
			0.4	84.16 ± 0.80	<b>83.96 ± 0.65</b>	83.80 ± 1.26	<b>83.43 ± 1.69</b>	83.50 ± 1.74	82.84 ± 2.33	3.27 ± 0.15	4.46 ± 0.45	2.42 ± 0.22
4		0	83.48 ± 4.32	82.97 ± 2.83	82.88 ± 3.22	<b>82.59 ± 2.86</b>	82.54 ± 3.56	82.38 ± 2.60	7.96 ± 0.28	51.84 ± 11.70	21.03 ± 4.44	
		0.1	83.76 ± 1.04	83.73 ± 1.22	83.63 ± 1.51	<b>83.30 ± 1.86</b>	83.34 ± 2.08	82.11 ± 2.53	4.15 ± 0.34	6.84 ± 0.66	3.35 ± 0.43	
		0.4	84.26 ± 0.11	84.23 ± 0.67	83.93 ± 1.10	<b>83.86 ± 1.43</b>	83.13 ± 1.18	82.44 ± 1.91	3.43 ± 0.28	5.02 ± 0.46	2.67 ± 0.21	
		0	82.87 ± 1.47	<b>82.42 ± 4.03</b>	<b>82.42 ± 4.21</b>	<b>82.08 ± 3.79</b>	<b>81.98 ± 2.13</b>	80.86 ± 2.53	8.08 ± 0.24	54.59 ± 4.36	21.65 ± 1.69	
		0.1	83.76 ± 1.01	<b>83.20 ± 1.41</b>	<b>83.17 ± 2.12</b>	<b>83.14 ± 2.36</b>	<b>83.10 ± 2.11</b>	82.44 ± 2.68	4.47 ± 0.35	7.49 ± 0.96	3.80 ± 0.41	
		0.4	84.96 ± 0.74	<b>84.36 ± 0.47</b>	<b>84.26 ± 1.90</b>	<b>84.13 ± 1.24</b>	<b>84.06 ± 1.01</b>	82.48 ± 3.04	3.29 ± 0.15	4.29 ± 0.22	2.41 ± 0.09	

In Table 6.12 the results of comparison between Cage-MetaCombiner and EvABCD are reported [54]. The EvABCD algorithm is a technique for classification of the behavior profiles of users. As Cage-MetaCombiner, it learns different behaviors from training data. For a different profile, it builds one or more prototypes computing the frequency of all the sequences of commands with a defined length. Moreover, it updates the models in an incremental way. The proposed algorithm performs better than EvABCD in most cases. By using Cage-MetaCombiner, the results for a different number of commands extracted do not influence the accuracy much, while the EvABCD algorithm improves its accuracy when using 500 commands.

**Table 6.12.** Comparison of the Cage-MetaCombiner vs the EvABCD algorithm for the Unix dataset (accuracy rate).

Commands	Sequence	Cage-Combiner	EvABCD
100	3	85.73	64.90
	4	83.86	64.50
	5	83.96	67.90
	6	85.06	64.30
500	3	84.39	59.50
	4	83.86	59.20
	5	83.76	66.70
	6	83.76	70.80



## 6.4 Experimental investigation on specialized ensemble approach

In this Section, the preliminary experiments performed to evaluate the effectiveness of the ensembles of specialized detectors are illustrated. To this aim, a comparison with other related approaches is presented using the well-known KDD dataset along with an analysis of the performance of the specialized ensembles on a really interesting dataset in the field of intrusion detection, presented in the Section 6.5.3, which overcomes the drawbacks of the KDD dataset.

### 6.4.1 Experiments on KDD 99 and ISCX IDS.

To evaluate the system proposed, the algorithm processes two datasets: the first, the KDD Cup 1999 is one of the most used in this field, but present some drawbacks, remarked in Section 6.5.3. However, it is included in the experiments for the sake of comparison with other similar approaches using this dataset. The details of connections and attacks are presented in the Section 6.1. The second dataset is ISCX IDS, also described in the Section 6.1.

Among the many metrics for evaluating classifier systems, the results are presented with the comparison of recall and precision, because they give an idea of the capacity of the system in individuating the attacks and in reducing the number of false alarms; indeed, recall represents the proportion of correctly predicted attack cases to the actual size of the attack class (a value of 100% indicates the detection of all the attacks, however, it can individuate also a large number of false attacks); precision represents the proportion of attack cases that were correctly predicted relative to the predicted size of the attack class (a value of 100% indicates that no false alarms were signaled, however a large number of alarms could be not detected).

In order to analyze the behavior of the approach in recognizing the minority classes, i.e., the attacks, the approach is compared with the one presented in [5], for the KDD Cup dataset. The authors propose a boosting approach, named Greedy-Boost, which builds an ensemble of classifier based on a linear combination of models, specifically designed to operate for the intrusion detection domain. The main idea of that algorithm is to extend the boosting process maintaining the models that behave better on the examples badly predicted in the previous round of the boosting algorithm (while the classical boosting algorithm adjusts only the weights and not the models).

Differently from the previous experiment done with this dataset 6.4.1, in this case only the specialized version of the ensemble is used. The main difference is the way how ensemble is composed. In this case CAGE-MetaCombiner builds an ensemble in which each model is trained to detect one type of attacks. In the previous case, all classifiers are able to detect all types of attack but with different results.

In Table 6.13, the results of the comparison between CAGE-MetaCombiner and the Greedy-Boost algorithm on the KDDCup 99 dataset are reported (precision and recall are reported for all the classes). It is evident that the approach performs better both for the precision and the recall measure when considering the minority classes, i.e. the attacks, and it is comparable for the other classes.

**Table 6.13.** Precision for different strategies for the KDD Cup dataset. In the first column, it is reported the class distribution for the dataset.

	Class Distribution	Precision	
		Greedy-Boost	CMC specialized
DoS	79.28%	100.0	99.99
Normal	19.86%	99.1	99.83
Probe	0.84%	99.0	99.43
R2L	0.023 %	93.2	98.29
U2R	1.06E-3 %	88.5	91.07

**Table 6.14.** Recall for different strategies for the KDD Cup dataset. In the first column, it is reported the class distribution for the dataset.

	Class Distribution	Recall	
		Greedy-Boost	CMC specialized
DoS	79.28%	100.0	99.99
Normal	19.86%	100.0	99.97
Probe	0.84%	97.1	97.31
R2L	0.023 %	71.9	86.43
U2R	1.06E-3 %	44.2	74.54

As for the ISCX dataset, in this case the goal is to verify the performance of the algorithm, trying to understand the benefits of using specialized classifier. It is worth to remember that an intrusion detection algorithm must be able to detect all the types of attack. Therefore, the labelled tuples of the dataset are randomly sampled and used as training and validation datasets to train one ensemble for each class; for the normal class, a completely random subsample of the dataset is extracted; for a given attack, a subsample having 50% of normal connections and 50% of the attack is extracted. The overall sample extracted represents 2% of the entire dataset. Finally, the entire dataset is classified with a weighted combination of the ensembles, as illustrated in Section 5.4.

In Table 6.15 are reported precision and recall for the CAGE-MetaCombiner algorithm, for the case of using specialized ensembles and for the case of using not specialized ensemble. The specialized ensembles behave better in terms of precision for all the types of attack (every day presents a different type of attack) while, as for the precision metric, the two approaches are comparable, with the exception of day 4, in which the behavior of the specialized ensembles is worst; this could be due to the nature of the attacks: indeed, most connections are related to HTTP DDoS attacks and these are quite close to normal connections resulting in increasing of the false positive rate.

**Table 6.15.** Precision and Recall for different strategies (with and without specialized ensemble) for the ISCX dataset.

	Precision		Recall	
	non specialized	specialized	non specialized	specialized
Day 2	96.12	99.47	99.95	100.00
Day 3	56.67	67.00	92.56	91.82
Day 4	42.24	54.88	81.63	67.55
Day 5	91.50	96.89	97.83	97.49
Day 7	96.58	97.35	99.94	100.00

## 6.5 Results with drift detection

Two set of experiments were performed, by using an artificial dataset and an intrusion detection dataset, in order to verify the goodness of the approach in a stream environment with the detection of drifts. In the next two subsections, the parameters used in the experiments and the results are illustrated.

### 6.5.1 Parameter settings

Among the many metrics for evaluating classifier systems, in this Section the results are reported in term of AUC. The AUC metric is the value of the area under the ROC curve. The ROC curve is computed comparing the false positive rate and the true positive rate. The first term measures the capacity to correctly detect attacks (i.e. recall). The second term measures the rate between the false alarm signaled above all normal connections processed. Computing the area, we have a number to describe the goodness of classifier. An AUC close to 1 means an optimal recognition rate.

The different classifiers composing the ensemble are trained on the same training set. In practice, for each window of the stream, 50% is used to train the base classifiers, the remaining 50% is used as validation set. The validation part is processed by the evolutionary algorithm to evolve the combination function. The maximum number of classifiers is fixed to 20 and the number of classifiers composing the ensemble are 10.

The CMC uses many learner as base classifiers, then it makes a selection using a strategy chosen among best, old and wheel selection.

The algorithms used as base classifiers in the experiments are based on the WEKA implementation<sup>10</sup> and are listed in the following: J48 (decision trees), JRIP rule learner (Ripper rule learning algorithm), NBTree (Naive Bayes tree), Naive Bayes, 1R classifier, logistic model trees, logistic regression, decision stumps and 1BK (k-nearest neighbor algorithm).

### 6.5.2 Experiments conducted on the artificial datasets

In order to evaluate the effectiveness of the replacement strategies, of the drift detection techniques used and in general of the approach, the experiments are conducted by varying the size of the evaluation window of the stream. In addition, the proposed approach is compared with the well-known incremental algorithm HoeffdingTree using the same drift detection algorithm and with a boosted version of the HoeffdingTree. Moreover, considering that the window size is the number of records used to train the base classifiers and to build the base classifier; the metrics of recall, precision and AUC are computed for each window and averaged over all the windows.

In all the tables of this Section and of the next one, the values that **are not** statistically significant different using the Friedman test (this choice is motivated by the observation that most of the differences are significative) are reported in bold. The

<sup>10</sup> <http://www.cs.waikato.ac.nz/ml/weka>

critical value of the Friedman test [32] is obtained from a chi-square distribution with two degree of freedom and a significancy level of 5%. The Friedman test is a non-parametric statistical test and it is used to detect differences across multiple test. The null hypothesis of this test is that the median value of all the populations is equal.

In Tables 6.16, 6.17 and 6.18, are shown respectively the precision, the recall and the AUC metrics concerning the HoeffdingTree algorithm (classic and boosted version) and the different replacement strategies used for CAGE-MetaCombiner: old, best and wheel. The experiments show that the size of the evaluation window does not affect significantly the performance of the algorithms. The CAGE-MetaCombiner approach is slightly better than the others and the better strategy of replacement is that replaces the worst trees with the best ones. This behavior can be referred to nature of the generated dataset. The same conclusions can be drawn for the AUC metric.

**Table 6.16.** Precision for CMC and HoeffdingTree (classic and boosted version) with HyperPlane dataset.

	Precision			
	1k	2k	5k	10k
	STEPD			
HT boosted	87.66 ± 2.12	87.66 ± 2.93	87.80 ± 1.97	87.97 ± 0.68
HoeffdingTree	87.48 ± 1.75	87.56 ± 0.27	87.64 ± 2.97	87.86 ± 2.81
CMC wheel	87.68 ± 1.52	87.87 ± 2.61	87.39 ± 1.26	87.49 ± 2.98
CMC best	87.68 ± 2.72	90.01 ± 0.81	93.28 ± 0.74	85.52 ± 0.99
CMC old	87.47 ± 2.61	89.78 ± 1.78	90.28 ± 2.35	85.55 ± 1.27
	ADWIN			
HT boosted	84.83 ± 0.05	85.00 ± 1.54	83.52 ± 0.33	83.52 ± 0.78
HoeffdingTree	86.44 ± 0.46	85.99 ± 0.07	86.25 ± 0.06	85.41 ± 0.59
CMC wheel	87.68 ± 2.46	85.48 ± 2.19	83.83 ± 0.92	84.72 ± 1.38
CMC best	87.68 ± 2.53	90.01 ± 0.41	91.92 ± 2.71	85.52 ± 0.21
CMC old	87.13 ± 1.71	89.29 ± 1.41	90.63 ± 1.77	84.23 ± 2.06

**Table 6.17.** Recall for CMC and HoeffdingTree (classic and boosted version) with HyperPlane dataset.

	Recall			
	1k	2k	5k	10k
	STEPD			
HT boosted	87.30 ± 2.69	87.37 ± 0.85	87.53 ± 2.07	87.59 ± 0.58
HoeffdingTree	87.25 ± 0.50	87.51 ± 2.66	87.55 ± 2.90	87.82 ± 1.04
CMC wheel	87.33 ± 0.32	87.48 ± 0.71	88.59 ± 1.17	87.98 ± 1.17
CMC best	91.32 ± 1.31	91.40 ± 2.67	85.86 ± 0.69	87.55 ± 0.15
CMC old	90.56 ± 2.77	89.01 ± 2.89	85.48 ± 0.83	87.32 ± 0.43
	ADWIN			
HT boosted	84.11 ± 1.40	84.16 ± 2.46	83.47 ± 1.85	83.77 ± 2.66
HoeffdingTree	86.48 ± 1.62	86.39 ± 1.62	85.45 ± 2.94	84.67 ± 1.70
CMC wheel	91.32 ± 0.84	85.36 ± 1.72	84.41 ± 1.59	83.79 ± 1.35
CMC best	91.32 ± 1.55	91.40 ± 1.16	91.66 ± 1.06	87.55 ± 1.92
CMC old	89.72 ± 2.75	89.80 ± 2.76	89.66 ± 2.13	85.55 ± 2.48

### 6.5.3 Experiments conducted on the real dataset

Generally, in the intrusion detection datasets, most of the instances represent normal connections, while the attacks represent the minority classes, often with different

**Table 6.18.** AUC metric for CMC and HoeffdingTree (classic and boosted version) with HyperPlane dataset.

	AUC			
	1k	2k	5k	10k
STEPD				
HT boosted	0.87 ± .029	0.88 ± .013	0.88 ± .012	0.88 ± .025
HoeffdingTree	0.87 ± .010	0.88 ± .022	0.88 ± .015	0.88 ± .001
CMC wheel	0.89 ± .019	0.89 ± .023	0.90 ± .002	0.89 ± .018
CMC best	0.89 ± .003	0.91 ± .001	0.90 ± .006	0.86 ± .019
CMC old	0.89 ± .013	0.90 ± .012	0.89 ± .004	0.86 ± .025
ADWIN				
HT boosted	0.86 ± .014	0.84 ± .010	0.83 ± .015	0.83 ± .029
HoeffdingTree	0.86 ± .078	0.86 ± .029	0.86 ± .007	0.85 ± .025
CMC wheel	0.89 ± .030	0.85 ± .010	0.84 ± .025	0.82 ± .015
CMC best	0.89 ± .016	0.91 ± .014	0.92 ± .026	0.86 ± .028
CMC old	0.88 ± .094	0.90 ± .012	0.92 ± .013	0.85 ± .022

orders of magnitude. Therefore, the ISCX dataset, described in the Subsection 6.1, which is representative of this behavior, was used for the experiments of this Section.

Analyzing Table 6.3 and the relative dataset, it is evident that the attacks are grouped in a small range of windows and, for different days, different kinds of attack can be observed. It worth to notice that drifts (changes in the data) can be detected when a new type of attack appears for the first time. Therefore, this dataset is appropriate in order to test the CMC approach.

In Tables 6.19, 6.20 and 6.21 the experimental results for the ISCX dataset are shown and the behavior of the approach is examined with the different replacement strategies.

In this case, the wheel selection strategy has the better performance for both the precision and recall metrics. The same behavior is observable in Table 6.21 for the AUC metric.

**Table 6.19.** Precision for different replacement strategies for CMC (ISCX dataset).

	Precision			
	1k	2k	5k	10k
STEPD				
CMC wheel	83.46 ± 0.29	89.40 ± 0.31	87.75 ± 2.24	89.97 ± 0.11
CMC best	79.19 ± 0.19	82.31 ± 2.65	74.30 ± 0.36	84.67 ± 2.29
CMC old	70.99 ± 0.95	75.98 ± 0.32	75.25 ± 0.59	78.98 ± 0.29
ADWIN				
CMC wheel	83.46 ± 0.81	87.59 ± 1.07	92.42 ± 0.77	88.28 ± 2.71
CMC best	57.98 ± 2.01	61.90 ± 0.79	65.90 ± 0.15	58.67 ± 1.92
CMC old	56.75 ± 2.73	63.65 ± 0.12	63.83 ± 1.99	59.80 ± 2.67

**Table 6.20.** Recall for different replacement strategies for CMC (ISCX dataset).

	Recall			
	1k	2k	5k	10k
STEPD				
CMC wheel	88.39 ± 2.75	85.47 ± 1.62	85.10 ± 2.71	80.41 ± 2.38
CMC best	65.98 ± 2.51	66.56 ± 2.40	82.85 ± 0.33	71.70 ± 2.96
CMC old	59.34 ± 1.63	62.35 ± 0.26	63.35 ± 1.09	67.98 ± 0.16
ADWIN				
CMC wheel	88.39 ± 2.51	88.25 ± 2.38	82.44 ± 2.44	80.79 ± 0.08
CMC best	67.44 ± 1.82	67.62 ± 0.60	73.90 ± 1.56	62.82 ± 0.69
CMC old	63.54 ± 2.82	64.34 ± 2.12	64.90 ± 0.71	61.98 ± 2.41

**Table 6.21.** AUC metric for different replacement strategies for CMC (ISCX dataset).

	AUC			
	1k	2k	5k	10k
STEPD				
CMC wheel	0.84 ± 0.019	0.86 ± 0.018	0.87 ± 0.007	0.87 ± 0.018
CMC best	0.72 ± 0.029	0.79 ± 0.016	0.86 ± 0.013	0.84 ± 0.008
CMC old	0.59 ± 0.009	0.60 ± 0.002	0.60 ± 0.007	0.62 ± 0.017
ADWIN				
CMC wheel	0.84 ± 0.014	0.87 ± 0.017	0.89 ± 0.007	0.89 ± 0.002
CMC best	0.51 ± 0.026	0.51 ± 0.018	0.55 ± 0.019	0.52 ± 0.015
CMC old	0.53 ± 0.003	0.53 ± 0.015	0.53 ± 0.001	0.52 ± 0.017

In Tables 6.22, 6.23 and 6.24 are shown respectively the recall and precision and the AUC metrics concerning the comparison between the HoeffdingTree algorithm (classic and boosted version) and the CMC approach with the wheel selection strategy. The choice of using ADWIN as drift detection algorithm is motivated by the consideration that it has equivalent performance (better in some cases) in comparison with STEPDP. Furthermore, considering only the AUC metric, ADWIN has a slight advantage on the performance of the STEPDP algorithm for the ISCX dataset. Note that, the HoeffdingTree algorithm updates more frequently its model, and therefore, also for small windows, it can improve quickly the predictive capacity. However, the CMC approach has performance close to the HoeffdingTree for small windows and it performs sensibly better when the window size grows. Furthermore, comparing the performance for both the version of the HoeffdingTree, a performance degradation of the ensemble-based algorithm can be observed. This behavior does not affect CMC, probably because when a drift is detected, it updates/replaces the models and re-weights the classifiers (by recomputing the combination function).

**Table 6.22.** Precision for the comparison among CMC and the Hoeffding tree (classical and boosted version) on ISCX dataset.

	Precision			
	1k	2k	5k	10k
ADWIN				
CMC wheel	83.46 ± 0.28	87.59 ± 0.03	92.42 ± 2.20	88.28 ± 2.69
HT boosted	84.72 ± 2.67	81.79 ± 2.82	79.50 ± 0.14	75.29 ± 2.82
HoeffdingTree	89.22 ± 1.67	87.51 ± 2.23	87.23 ± 1.80	87.48 ± 1.51

**Table 6.23.** Recall for the comparison among CMC and the Hoeffding tree (classical and boosted version) on ISCX dataset.

	Recall			
	1k	2k	5k	10k
ADWIN				
CMC wheel	88.39 ± 0.69	88.25 ± 1.08	82.44 ± 0.69	80.79 ± 0.26
HT boosted	85.20 ± 1.04	81.35 ± 2.14	70.62 ± 1.15	58.72 ± 1.90
HoeffdingTree	92.66 ± 0.92	87.76 ± 2.86	79.69 ± 0.53	75.62 ± 2.10

**Table 6.24.** AUC metric for the comparison among CMC and the Hoeffding tree (classical and boosted version) on ISCX dataset.

	AUC			
	1k	2k	5k	10k
	ADWIN			
CMC wheel	0.84 ± .022	0.87 ± .003	0.89 ± .001	0.89 ± .007
HT boosted	0.82 ± .021	0.79 ± .007	0.75 ± .009	0.69 ± .003
HoeffdingTree	0.89 ± .014	0.88 ± .001	0.85 ± .012	0.83 ± .013



## Conclusion and future works

In this thesis, a general architecture based on the ensemble paradigm is proposed. The main aim of work is to design a distributed system for environment in which performance and scalability are the most important aspect to be considered. Two scenarios in the cyber security field are illustrated: a system to identify user profiles and an innovative architecture for a distributed intrusion detection system. In both cases, the classifiers must have the ability to handle unbalanced data, missing data and, considering streaming data, the ability to detect different types of drifts.

The first scenario is a system, which permits to divide users in groups with the same behavior and the same weaknesses. This task is very important in modern architecture because the complexity makes difficult to detect attacks started from legitimate users. An architecture to solve this problem is proposed and the experimental results have proved the effectiveness of the approach. Moreover, the system is able to works with large logs.

Another scenario considered is a system designed to detect network intrusions. The flexibility and versatility of the ensemble approach is easy to apply in the research theme of intrusion detection. The goal is to design a distributed architecture for an IDS. First of all, many data mining algorithms that present an approach suitable to network intrusion detection to take advantage of modern parallel/distributed and cloud environments are reviewed. Then, all components of the system are defined considering two types of input: data generated from "static" applications and data generated in a streaming environment.

These scenarios have highlighted problems such as unbalanced data, missing data, changes in data (drift), etc. Indeed, in this particular domain, datasets often have different number of features and each attribute could have different importance and costs. Furthermore, some data source could be not present but the system must work anyway. Therefore, it would be really unlikely a single classification algorithm will perform well for all the datasets, especially in presence of changes and with constraints of real time and scalability.

The solutions proposed by this thesis to these problems is a general framework based on ensemble paradigm and evolutionary algorithm. The framework is described in term of problems addressed and an architecture is defined for each ap-

plication considered. The experimental results are conducted to test the effectiveness of the approach and they are divided in several test suites.

The first test suite analyzes the performance of the approach with unbalanced data. The system based on an ensemble model evolves a combiner function, which does not need additional phases of training, after the heterogeneous classifiers composing the ensemble are trained. The experiments showed that the proposed system improves or is comparable to the performance of state-of-the-art approaches for combining ensemble, by using a smaller number of models.

The second test suite concerns the missing data problem. A meta-ensemble-based framework for classifying datasets in the cyber security domain and a real scenario concerning the segmentation of the users of an e-payment system, which illustrates the real applicability of the approach, is realized. The main advantages of the framework are its capacity in handling groups of missing feature and the possibility of operating in an incremental way without the need for re-training on the original data. In addition, in the case of missing features, it works better than other similar approaches when an entire group of features is missing.

The third test suite is dedicated to test the ability to handle very difficult problems in the field of intrusion detection, the system is extended using specialized models as base classifiers for the ensemble. A distributed framework for the problem of intrusion detection has been presented and the different components were illustrated. In particular, the meta-ensemble engine used to classify the attacks and the normal connections, based on a non-trainable function evolved by the GP tool, is test on well-known datasets.

The experiments conducted on the KDD dataset and on a more up-to-date dataset demonstrate that the framework is able to cope with the intrusion detection task even in the case of really unbalanced classes, by exploiting the advantages of the specialized classifiers. The algorithm succeeds in minimizing the false alarms and generally recognizes well the attacks.

Finally, the last tests are analyzed the performance of the proposed approach with the streaming data and a drift detection module is added to the general architecture. As for the previous cases, the system evolves a combiner function afterwards the heterogeneous classifiers composing the ensemble are trained. The framework includes a drift detection function to detect changes in the data and a strategy for replacing classifiers, which permits to build the ensemble in an incremental way. The experiments showed that the framework is apt to cope with large streams of normal connections and attacks and that the new proposed selection strategy obtains encouraging results in comparison with classical replacement strategies.

The design of the framework has highlighted some considerations that have a key role for any cyber security applications. The distributed computing as well as the sharing of the models are two approaches that permit to build a very efficient system able to satisfy the hard requirements of modern applications. Moreover, an in depth analysis on real data has high relevance to understand the ability of an application to solve a specific problem.

The experiments has also shown how the framework is flexible to handle many different problems, and the phase of optimization and setting tuning for each dataset is the proof that some improvements are feasible.

The time and space requirements of this research have been limited by the appropriate scope of research for the thesis, but a number of possible future topics that would be interesting to explore can be identified.

As example, a real environment can be used to test the framework. Regarding the data dimension, an interesting field of research is to understand how the framework works with big data. Although, the framework has a distributed implementation, working with big data has new issues to address.

Another research field is the use of modern infrastructure of cloud computing. Map-reduce based technologies are an interesting field, and the extension of the framework to work with these approaches can be seen as an interesting improvements.

The integration of framework in popular platform of machine learning makes easy to work with streaming data.

In the field of intrusion detection, many attacks are new, i.e. never met before; however the classification algorithms used in the framework require that the attacks are known in advance. Therefore, the usage of unsupervised approaches could address this issue, but their integration in the framework must be carefully analyzed.



---

## References

1. Acosta-Mendoza, N., Morales-Reyes, A., Escalante, H.J., Gago-Alonso, A.: Learning to assemble classifiers via genetic programming. *IJPRAI* **28**(7) (2014)
2. Agrawal, R., Gehrke, J., Gunopulos, D., Raghavan, P.: Automatic subspace clustering of high dimensional data. *Data Mining and Knowledge Discovery* **11**(1), 5–33 (2005)
3. Alshwabkeh, M., Jang, B., Kaeli, D.: Accelerating the local outlier factor algorithm on a gpu for intrusion detection systems. In: *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units, GPGPU-3*, pp. 104–110. ACM, New York, NY, USA (2010). DOI 10.1145/1735688.1735707. URL <http://doi.acm.org/10.1145/1735688.1735707>
4. Ariu, D., Tronci, R., Giacinto, G.: Hmmpayl: An intrusion detection system based on hidden markov models. *computers & security* **30**(4), 221–241 (2011)
5. Bahri, E., Harbi, N., Huu, H.N.: Approach based ensemble methods for better and faster intrusion detection. In: *4th Int. Conf. on Computational Intelligence in Security for Information Systems*, pp. 17–24. Springer (June 8-10, 2011)
6. Bandre, S., Nandimath, J.: Design consideration of network intrusion detection system using hadoop and gpgpu. In: *Pervasive Computing (ICPC), 2015 International Conference on*, pp. 1–6 (2015). DOI 10.1109/PERVASIVE.2015.7087201
7. Bellekens, X.J.A., Tachtatzis, C., Atkinson, R.C., Renfrew, C., Kirkham, T.: A highly-efficient memory-compression scheme for gpu-accelerated intrusion detection systems. In: *Proceedings of the 7th International Conference on Security of Information and Networks, SIN '14*, pp. 302:302–302:309. ACM, New York, NY, USA (2014). DOI 10.1145/2659651.2659723. URL <http://doi.acm.org/10.1145/2659651.2659723>
8. Bhuyan, M., Bhattacharyya, D., Kalita, J.: Network anomaly detection: Methods, systems and tools. *Communications Surveys Tutorials, IEEE* **16**(1), 303–336 (2014). DOI 10.1109/SURV.2013.052213.00046
9. Bhuyan, M.H., Bhattacharyya, D., Kalita, J.: Surveying port scans and their detection methodologies. *Comput. J.* **54**(10), 1565–1581 (2011). DOI 10.1093/comjnl/bxr035. URL <http://dx.doi.org/10.1093/comjnl/bxr035>
10. Bhuyan, M.H., Bhattacharyya, D.K., Kalita, J.K.: Nado: Network anomaly detection using outlier approach. In: *Proceedings of the 2011 International Conference on Communication, Computing & Security, ICCCS '11*, pp. 531–536. ACM, New York, NY, USA (2011). DOI 10.1145/1947940.1948050. URL <http://doi.acm.org/10.1145/1947940.1948050>

11. Bhuyan, M.H., Bhattacharyya, D.K., Kalita, J.K.: Survey on incremental approaches for network anomaly detection. *International Journal of Communication Networks and Information Security (IJCNIS)* **3**(3) (2011)
12. Bifet, A., Gavaldà, R.: Learning from time-changing data with adaptive windowing. In: *SDM*, vol. 7, p. 2007. SIAM (2007)
13. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM* **13**(7), 422–426 (1970)
14. Boggs, N., Hiremagalore, S., Stavrou, A., Stolfo, S.: Cross-domain collaborative anomaly detection: So far yet so close. In: R. Sommer, D. Balzarotti, G. Maier (eds.) *Recent Advances in Intrusion Detection, Lecture Notes in Computer Science*, vol. 6961, pp. 142–160. Springer Berlin Heidelberg (2011). DOI 10.1007/978-3-642-23644-0\_8. URL [http://dx.doi.org/10.1007/978-3-642-23644-0\\_8](http://dx.doi.org/10.1007/978-3-642-23644-0_8)
15. Brahmi, I., Ben Yahia, S., Aouadi, H., Poncelet, P.: Towards a multiagent-based distributed intrusion detection system using data mining approaches. In: *Proceedings of the 7th International Conference on Agents and Data Mining Interaction, ADMI'11*, pp. 173–194. Springer-Verlag, Berlin, Heidelberg (2012)
16. Brameier, M., Banzhaf, W.: Evolving teams of predictors with linear genetic programming. *Genetic Programming and Evolvable Machines* **2**(4), 381–407 (2001)
17. Breiman, L.: Bagging predictors. *Machine Learning* **24**(2), 123–140 (1996)
18. Bukhtoyarov, V., Zhukov, V.: Ensemble-distributed approach in classification problem solution for intrusion detection systems. In: E. Corchado, J.A. Lozano, H. Quintián, H. Yin (eds.) *Intelligent Data Engineering and Automated Learning, IDEAL 2014, Lecture Notes in Computer Science*, vol. 8669, pp. 255–265. Springer International Publishing (2014). DOI 10.1007/978-3-319-10840-7\_32. URL [http://dx.doi.org/10.1007/978-3-319-10840-7\\_32](http://dx.doi.org/10.1007/978-3-319-10840-7_32)
19. Bukhtoyarov, V.V., Semenkina, O.E.: Comprehensive evolutionary approach for neural network ensemble automatic design. In: *Evolutionary Computation (CEC), 2010 IEEE Congress on*, pp. 1–6 (2010)
20. Campos, M., Milenova, B.: Creation and deployment of data mining-based intrusion detection systems in oracle database 10g. In: *Machine Learning and Applications, 2005. Proceedings. Fourth International Conference on*, pp. 8 pp.– (2005)
21. Casas, P., Mazel, J., Owezarski, P.: Unsupervised network intrusion detection systems: Detecting the unknown without knowledge. *Computer Communications* **35**(7), 772 – 783 (2012)
22. CERT Australia: Cyber crime and security survey report. Tech. rep. (2012)
23. Chawla, N., Sylvester, J.: Exploiting diversity in ensembles: Improving the performance on unbalanced datasets. In: *Multiple Classifier Systems, 7th International Workshop*, pp. 397–406. Springer (2007)
24. Chen, H., Du, Y., Jiang, K.: Classification of incomplete data using classifier ensembles. In: *Systems and Informatics (ICSAI), 2012 International Conference on*, pp. 2229–2232 (2012)
25. Corona, I., Giacinto, G., Mazzariello, C., Roli, F., Sansone, C.: Information fusion for computer security: State of the art and open issues. *Information Fusion* **10**(4), 274 – 284 (2009). DOI <http://dx.doi.org/10.1016/j.inffus.2009.03.001>. URL <http://www.sciencedirect.com/science/article/pii/S156625350900030X>. Special Issue on Information Fusion in Computer Security
26. Cretu, G., Stavrou, A., Locasto, M., Stolfo, S., Keromytis, A.: Casting out demons: Sanitizing training data for anomaly sensors. In: *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pp. 81–95 (2008)

27. Cretu, G.F., Stavrou, A., Stolfo, S.J., Keromytis, A.D.: Data sanitization: Improving the forensic utility of anomaly detection systems. In: Proceedings of the 3rd Workshop on on Hot Topics in System Dependability, HotDep'07. USENIX Association, Berkeley, CA, USA (2007). URL <http://dl.acm.org/citation.cfm?id=1323140.1323142>
28. Cretu-Ciocarlie, G.F., Stavrou, A., Locasto, M.E., Stolfo, S.J.: Adaptive anomaly detection via self-calibration and dynamic updating. In: Recent Advances in Intrusion Detection, pp. 41–60. Springer (2009)
29. Czarnowski, I., Jędrzejowicz, P.: Ensemble classifier for mining data streams. *Procedia Computer Science* **35**, 397–406 (2014)
30. Dasarathy, B.V.: Intrusion detection. *Information Fusion* **4**(4), 243–245 (2003)
31. Davis, J.J., Clark, A.J.: Data preprocessing for anomaly based network intrusion detection: A review. *Computers & Security* **30**(6-7), 353 – 375 (2011). DOI <http://dx.doi.org/10.1016/j.cose.2011.05.008>. URL <http://www.sciencedirect.com/science/article/pii/S0167404811000691>
32. Demsar, J.: Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research* **7**, 1–30 (2006)
33. Dua, S., Du, X.: *Data Mining and Machine Learning in Cybersecurity*. CRC Press (2011)
34. Ester, M., Kriegel, H.P., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: *Kdd*, vol. 96, pp. 226–231 (1996)
35. Folino, G., Pisani, F.: Evolving meta-ensemble of classifiers for handling incomplete and unbalanced datasets in the cyber security domain. *Applied Soft Computing* **47**, 179–190 (2016)
36. Folino, G., Pizzuti, C., Spezzano, G.: A scalable cellular implementation of parallel genetic programming. *IEEE Transactions on Evolutionary Computation* **7**(1), 37–53 (2003)
37. Folino, G., Pizzuti, C., Spezzano, G.: Training Distributed GP Ensemble With a Selective Algorithm Based on Clustering and Pruning for Pattern Classification. *IEEE Trans. Evolutionary Computation* **12**(4), 458–468 (2008)
38. Folino, G., Pizzuti, C., Spezzano, G.: An ensemble-based evolutionary framework for coping with distributed intrusion detection. *Genetic Programming and Evolvable Machines* **11**(2), 131–146 (2010). DOI [10.1007/s10710-010-9101-6](https://doi.org/10.1007/s10710-010-9101-6). URL <http://dx.doi.org/10.1007/s10710-010-9101-6>
39. Folino, G., Sabatino, P.: Ensemble based collaborative and distributed intrusion detection systems: A survey. *J. Netw. Comput. Appl.* **66**(C), 1–16 (2016). DOI [10.1016/j.jnca.2016.03.011](https://doi.org/10.1016/j.jnca.2016.03.011). URL <http://dx.doi.org/10.1016/j.jnca.2016.03.011>
40. Folino, G., Sabatino, P., Pisani, F.S.: Combining ensemble of classifiers by using genetic programming for cyber security applications. In: *Applications of Evolutionary Computation - 19th European Conference, EvoApplications 2016, Porto, Portugal, 30 March - 1 April, 2016, Proceedings, Lecture Notes in Computer Science*. Springer International Publishing (2016)
41. Ford, D.: Application of thermodynamics to the reduction of data generated by a non-standard system. *arXiv preprint cond-mat/0402325* (2004)
42. Foremski, P., Callegari, C., Pagano, M.: Waterfall: Rapid Identification of IP Flows Using Cascade Classification. In: A. Kwiecień, P. Gaj, P. Stera (eds.) *Computer Networks, Communications in Computer and Information Science*, vol. 431, pp. 14–23. Springer International Publishing (2014). DOI [10.1007/978-3-319-07941-7\\_2](https://doi.org/10.1007/978-3-319-07941-7_2). URL [http://dx.doi.org/10.1007/978-3-319-07941-7\\_2](http://dx.doi.org/10.1007/978-3-319-07941-7_2)
43. Fred, A.L.N., Jain, A.K.: Combining multiple clusterings using evidence accumulation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **27**(6), 835–850 (2005)

44. Freund, Y., Shapire, R.: Experiments with a new boosting algorithm. In: Machine Learning, Proceedings of the Thirteenth International Conference (ICML '96), pp. 148–156. Morgan Kaufmann (1996)
45. Garcia-Teodoro, P., Diaz-Verdejo, J., Maciá-Fernández, G., Vázquez, E.: Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security* **28**(1-2), 18 – 28 (2009)
46. Gauch, S., Speretta, M., Chandramouli, A., Micarelli, A.: User profiles for personalized information access. In: P. Brusilovsky, A. Kobsa, W. Nejdl (eds.) *The Adaptive Web, Lecture Notes in Computer Science*, vol. 4321, pp. 54–89. Springer (2007)
47. Giacinto, G., Roli, F., Didaci, L.: Fusion of multiple classifiers for intrusion detection in computer networks. *Pattern recognition letters* **24**(12), 1795–1803 (2003)
48. Godoy, D., Amandi, A.: User profiling in personal information agents: A survey. *Knowl. Eng. Rev.* **20**(4), 329–361 (2005)
49. Greenberg, S.: Using unix: Collected traces of 168 users. In: Research Report 88/333/45. Department of Computer Science, University of Calgary, Calgary, Canada. (1988)
50. Hoque, N., Bhuyan, M.H., Baishya, R.C., Bhattacharyya, D.K., Kalita, J.K.: Network attacks: Taxonomy, tools and systems. *Journal of Network and Computer Applications* **40**(0), 307 – 324 (2014). DOI <http://dx.doi.org/10.1016/j.jnca.2013.08.001>
51. Hu, W., Gao, J., Wang, Y., Wu, O., Maybank, S.: Online adaboost-based parameterized methods for dynamic distributed network intrusion detection. *Cybernetics, IEEE Transactions on* **44**(1), 66–82 (2014)
52. Huang, S., Wang, B., Qiu, J., Yao, J., Wang, G., Yu, G.: Parallel ensemble of online sequential extreme learning machine based on MapReduce. *Neurocomputing* **174**, Part A, 352 – 367 (2016)
53. Hulten, G., Spencer, L., Domingos, P.: Mining time-changing data streams. In: Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 97–106. ACM (2001)
54. Iglesias, J.A., Angelov, P.P., Ledezma, A., Sanchis, A.: Creating evolving user behavior profiles automatically. *IEEE Trans. Knowl. Data Eng.* **24**(5), 854–867 (2012)
55. Jamshed, M.A., Lee, J., Moon, S., Yun, I., Kim, D., Lee, S., Yi, Y., Park, K.: Kargus: A highly-scalable software-based intrusion detection system. In: Proceedings of the 2012 ACM Conference on Computer and Communications Security, CCS '12, pp. 317–328. ACM, New York, NY, USA (2012). DOI [10.1145/2382196.2382232](https://doi.org/10.1145/2382196.2382232). URL <http://doi.acm.org/10.1145/2382196.2382232>
56. Jolliffe, I.: *Principal Component Analysis*. Springer Series in Statistics. Springer (2002). URL [http://books.google.it/books?id=\\_olByCrhjwIC](http://books.google.it/books?id=_olByCrhjwIC)
57. Jr., P.M.G., de Carvalho Santos, S.G.T., de Barros, R.S.M., Vieira, D.C.D.L.: A comparative study on concept drift detectors. *Expert Syst. Appl.* **41**(18), 8144–8156 (2014)
58. Kemmerer, R., Vigna, G.: Intrusion detection: a brief history and overview. *Computer* **35**(4), 27–30 (2002)
59. Kephart, J.O., Chess, D.M.: The vision of autonomic computing. *Computer* **36**(1), 41–50 (2003). DOI [10.1109/MC.2003.1160055](https://doi.org/10.1109/MC.2003.1160055). URL <http://dx.doi.org/10.1109/MC.2003.1160055>
60. Khasawneh, K., Ozsoy, M., Donovick, C., Abu-Ghazaleh, N., Ponomarev, D.: Ensemble learning for low-level hardware-supported malware detection. In: H. Bos, F. Monrose, G. Blanc (eds.) *Research in Attacks, Intrusions, and Defenses, Lecture Notes in Computer Science*, vol. 9404, pp. 3–25. Springer International Publishing (2015). DOI [10.1007/978-3-319-26362-5\\_1](https://doi.org/10.1007/978-3-319-26362-5_1)
61. Koza, J.R.: *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA (1992)



62. Kuncheva, L.: Combining pattern classifiers: methods and algorithms. John Wiley & Sons (2004)
63. Lai, H., Cai, S., Huang, H., Xie, J., Li, H.: A parallel intrusion detection system for high-speed networks. In: M. Jakobsson, M. Yung, J. Zhou (eds.) Applied Cryptography and Network Security, *Lecture Notes in Computer Science*, vol. 3089, pp. 439–451. Springer Berlin Heidelberg (2004)
64. Leung, K., Leckie, C.: Unsupervised anomaly detection in network intrusion detection using clusters. In: Proceedings of the Twenty-eighth Australasian Conference on Computer Science - Volume 38, ACSC '05, pp. 333–342. Australian Computer Society, Inc., Darlinghurst, Australia, Australia (2005). URL <http://dl.acm.org/citation.cfm?id=1082161.1082198>
65. Levey, A., Lindenbaum, M.: Sequential Karhunen-Loeve basis extraction and its application to images. *Image Processing, IEEE Transactions on* **9**(8), 1371–1374 (2000). DOI 10.1109/83.855432
66. Liang, N.Y., Huang, G.B., Saratchandran, P., Sundararajan, N.: A fast and accurate online sequential learning algorithm for feedforward networks. *Neural Networks, IEEE Transactions on* **17**(6), 1411–1423 (2006)
67. Little, R.J.A., Rubin, D.B.: Statistical Analysis with Missing Data. John Wiley & Sons, Inc., New York, NY, USA (1986)
68. Locasto, M.E., Parekh, J.J., Keromytis, A.D., Stolfo, S.J.: Towards collaborative security and p2p intrusion detection. In: Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC, pp. 333–339. IEEE (2005)
69. Luke, S., Panait, L.: A comparison of bloat control methods for genetic programming. *Evol. Comput.* **14**(3), 309–344 (2006)
70. McEachen, J.C., Kah, C.W.: An analysis of distributed sensor data aggregation for network intrusion detection. *Microprocessors and Microsystems* **31**(4), 263 – 272 (2007). Special Issue with selected papers from the 11th IEEE Symposium on Computers and Communications (ISCC 2006)
71. Nagesh, H.S., Goil, S., Choudhary, A.: A scalable parallel subspace clustering algorithm for massive data sets. In: Parallel Processing, 2000. Proceedings. 2000 International Conference on, pp. 477–484. IEEE (2000)
72. Newsome, J., Karp, B., Song, D.: Polygraph: automatically generating signatures for polymorphic worms. In: Security and Privacy, 2005 IEEE Symposium on, pp. 226–241 (2005). DOI 10.1109/SP.2005.15
73. Nishida, K., Yamauchi, K.: Detecting concept drift using statistical testing. In: Discovery Science, pp. 264–269. Springer (2007)
74. de Oliveira, D.F., Canuto, A.M.P., de Souto, M.C.P.: Use of multi-objective genetic algorithms to investigate the diversity/accuracy dilemma in heterogeneous ensembles. In: International Joint Conference on Neural Networks, pp. 2339–2346. IEEE (2009)
75. Parikh, D., Chen, T.: Data fusion and cost minimization for intrusion detection. *IEEE Transactions on Information Forensics and Security* **3**(3), 381–389 (2008)
76. Patel, K., Buddhadev, B.V.: Machine learning based research for network intrusion detection: A state-of-the-art. *International Journal of Information and Network Security (IJINS)* **3**(3) (2014). URL <http://iaesjournal.com/online/index.php/IJINS/article/view/6222>
77. Perdisci, R., Ariu, D., Giacinto, G.: Scalable fine-grained behavioral clustering of http-based malware. *Computer Networks* **57**(2), 487 – 500 (2013). DOI <http://dx.doi.org/10.1016/j.comnet.2012.06.022>. URL <http://www.sciencedirect.com/science/article/pii/S1389128612002678>. Botnet Activity: Analysis, Detection and Shutdown

78. Perdisci, R., Lee, W., Feamster, N.: Behavioral clustering of http-based malware and signature generation using malicious network traces. In: Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation, NSDI'10, pp. 26–26. USENIX Association, Berkeley, CA, USA (2010). URL <http://dl.acm.org/citation.cfm?id=1855711.1855737>
79. Pietruczuk, L., Rutkowski, L., Jaworski, M., Duda, P.: A method for automatic adjustment of ensemble size in stream data mining. In: Neural Networks (IJCNN), 2016 International Joint Conference on, pp. 9–15. IEEE (2016)
80. Polikar, R., DePasquale, J., Mohammed, H.S., Brown, G., Kuncheva, L.I.: Learn++.mf: A random subspace approach for the missing feature problem. *Pattern Recognition* **43**(11), 3817–3832 (2010)
81. Raut, A.S., Singh, K.R.: Anomaly based intrusion detection-a review. *Int. J. on Network Security* **5** (2014)
82. Schaelicke, L., Wheeler, K., Freeland, C.: Spanids: A scalable network intrusion detection loadbalancer. In: Proceedings of the 2Nd Conference on Computing Frontiers, CF '05, pp. 315–322. ACM, New York, NY, USA (2005)
83. Schapire, R.E.: The strength of weak learnability. *Machine Learning* **5**(2), 197–227 (1990)
84. Schapire, R.E.: Boosting a weak learning by majority. *Information and Computation* **121**(2), 256–285 (1995)
85. Shiravi, A., Shiravi, H., Tavallae, M., Ghorbani, A.A.: Toward developing a systematic approach to generate benchmark datasets for intrusion detection. *Computers & Security* **31**(3), 357 – 374 (2012). DOI <http://dx.doi.org/10.1016/j.cose.2011.12.012>. URL <http://www.sciencedirect.com/science/article/pii/S0167404811001672>
86. Sperotto, A., Schaffrath, G., Sadre, R., Morariu, C., Pras, A., Stiller, B.: An overview of ip flow-based intrusion detection. *Communications Surveys Tutorials, IEEE* **12**(3), 343–356 (2010). DOI 10.1109/SURV.2010.032210.00054
87. Stefano, C.D., Folino, G., Fontanella, F., di Freca, A.S.: Using bayesian networks for selecting classifiers in GP ensembles. *Information Sciences* **258**, 200–216 (2014)
88. Subrahmanian, V., Ovelgonne, M., Dumitras, T., Prakash, B.A.: *The Global Cyber-Vulnerability Report*, 1 edn. Springer (2015)
89. Sylvester, J., Chawla, N.V.: Evolutionary ensembles: Combining learning agents using genetic algorithms. In: AAI Workshop on Multiagent Learning, pp. 46–51 (2005)
90. Sylvester, J., Chawla, N.V.: Evolutionary ensemble creation and thinning. In: Proceedings of the International Joint Conference on Neural Networks, IJCNN 2006, pp. 5148–5155. IEEE (2006)
91. Tavallae, M., Stakhanova, N., Ghorbani, A.: Toward credible evaluation of anomaly-based intrusion-detection methods. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on* **40**(5), 516–524 (2010). DOI 10.1109/TSMCC.2010.2048428
92. Vasiliadis, G., Polychronakis, M., Ioannidis, S.: Midea: A multi-parallel intrusion detection architecture. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, CCS '11, pp. 297–308. ACM, New York, NY, USA (2011). DOI 10.1145/2046707.2046741. URL <http://doi.acm.org/10.1145/2046707.2046741>
93. Vasilomanolakis, E., Karuppayah, S., Mühlhäuser, M., Fischer, M.: Taxonomy and survey of collaborative intrusion detection. *ACM Comput. Surv.* **47**(4), 55:1–55:33 (2015)
94. Vokorokos, L., Ennert, M., Cajkovsky, M., Turinska, A.: A distributed network intrusion detection system architecture based on computer stations using gpgpu. In: *Intelligent*

- Engineering Systems (INES), 2013 IEEE 17th International Conference on, pp. 323–326 (2013). DOI 10.1109/INES.2013.6632834
95. Vokorokos, L., Ennert, M., Čajkovský, M., Raduřvský, J.: A survey of parallel intrusion detection on graphical processors. *Central European Journal of Computer Science* **4**(4), 222–230 (2014)
  96. Wang, K., Parekh, J.J., Stolfo, S.J.: Anagram: A content anomaly detector resistant to mimicry attack. In: *Recent Advances in Intrusion Detection*, pp. 226–248. Springer (2006)
  97. Wang, W., Guyet, T., Quiniou, R., Cordier, M.O., Maseglia, F., Zhang, X.: Autonomic intrusion detection: Adaptively detecting anomalies over unlabeled audit data streams in computer networks. *Knowledge-Based Systems* **70**(0), 103 – 117 (2014)
  98. Wang, Y., Gao, Y., Shen, R., Yang, F.: Selective ensemble approach for classification of datasets with incomplete values. In: *Foundations of Intelligent Systems*, pp. 281–286. Springer (2012)
  99. van Zadelhoff, M.: The biggest cybersecurity threats are inside your company. Digital article - Harvard Business Review (2016)
  100. Zhang, T., Ramakrishnan, R., Livny, M.: Birch: an efficient data clustering method for very large databases. In: *ACM SIGMOD Record*, vol. 25, pp. 103–114. ACM (1996)
  101. Zheng, K., Cai, Z., Zhang, X., Wang, Z., Yang, B.: Algorithms to speedup pattern matching for network intrusion detection systems. *Computer Communications* **62**, 47 – 58 (2015)
  102. Zhou, C.V., Leckie, C., Karunasekera, S.: Decentralized multi-dimensional alert correlation for collaborative intrusion detection. *Journal of Network and Computer Applications* **32**(5), 1106 – 1123 (2009). DOI <http://dx.doi.org/10.1016/j.jnca.2009.02.010>. Next Generation Content Networks
  103. Zhou, C.V., Leckie, C., Karunasekera, S.: A survey of coordinated attacks and collaborative intrusion detection. *Computers & Security* **29**(1), 124 – 140 (2010)