High performance big data analysis ; application to anomaly detection in the context of identity and access management

Mamadou Abdoulaye Diop

Thèse de doctorat

NNT : 2021UPASG100

université PARIS-SACLAY

Analyse haute performance de masses de données ; application à la détection d'anomalie dans le contexte de la gestion d'identité et d'accès

High performance big data analysis; application to anomaly detection in the context of identity and access management

Thèse de doctorat de l'université Paris-Saclay

École doctorale n° 580, Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat: Informatique
Unité de recherche : Université Paris-Saclay, UVSQ, LI PARAD, 78180, Saint-Quentin en Yvelines, France.
Référent : Université de Versailles -Saint-Quentin-en-Yvelines

Thèse présentée et soutenue à Paris-Saclay, le 10/12/2021, par

Mamadou Abdoulaye DIOP

Composition du Jury

Marc BABOULIN Professeur, Université de Paris Saclay, France	Président ou Présidente
Vassil ALEXANDROV Scientific Director, STFC Hartree Center, United Kingdom	Rapporteur & Examinateur / trice
Tetsuya SAKURAI Professor, University of Tsukuba, Japan	Rapporteur & Examinateur / trice
Leroy Anthony DRUMMOND Professor, University of California Berkeley, USA	Examinateur ou Examinatrice
Mathilde MOUGEOT Professeure, École Nationale Supérieure, France	Examinateur ou Examinatrice
Maxime HUGUES Docteur, Amazon Web Services, USA	Examinateur ou Examinatrice

Direction de la thèse

Nahid EMAD Professeure, Université de Paris Saclay, France	Directeur ou Directrice de thèse
Thierry WINTER CTO, Atos-Evidian, France	Co-Directeur ou co-directrice de thèse

Aujourd'hui, la majorité des entreprises et des sociétés utilisent des solutions numériques pour gérer leur patrimoine informationnel. Ces actifs sont une source de profit; les organisations comprennent donc l'importance stratégique de protéger leurs systèmes d'information. Elles utilisent des outils de cybersécurité pour se protéger contre toutes sortes de cyberattaques. Cependant, ces attaques ne cessent de prendre de l'ampleur car les cybercriminels ont compris l'aspect lucratif des données et élargissent continuellement leurs panels d'attaque.

Pendant longtemps, les spécialistes de la cybersécurité se sont concentrés sur la défense contre les menaces externes. Ils utilisaient des outils basés sur des bases de données de menaces connues contre les cyberattaques et surveillaient l'accès à leurs réseaux, serveurs et terminaux internes. Plus précisément, ils ont utilisé des solutions de gestion des identités et des accès (IAM) pour gérer l'accès aux actifs informationnels des organisations. Ces outils permettent de fournir et de contrôler l'accès aux informations et services critiques pour les personnes autorisées. Ils contribuent à réduire considérablement le risque de perte, d'utilisation abusive et de sabotage des données. Cependant, les solutions IAM ne peuvent pas identifier les utilisateurs ayant un accès légitime aux données qui utilisent leur droit d'accès de manière malveillante. Dans le domaine de la cybersécurité, ce problème est défini comme une menace interne. Dans un environnement d'entreprise, les insiders sont des employés de l'entreprise qui font un usage abusif de leurs droits d'accès. Un exemple typique d'activité d'insider de la part d'un individu est la vente d'informations commerciales à des organisations concurrentes en échange d'une compensation (par exemple, l'espionnage industriel).

Un autre exemple est le sabotage des outils propriétaires d'une organisation par un individu mandaté par une entreprise rivale. Selon IBM en 2015, 60% des cybermenaces étaient dues à des insiders (soit 44,5% d'insiders malveillants et 15,5% d'acteurs involontaires). En 2020, dans le rapport sur la menace des insiders, une enquête menée par les "insiders" de la cybersécurité a révélé que seuls 5% des professionnels de la cybersécurité interrogés se sentent invulnérables face aux menaces internes.

La recrudescence de ce type de menace a fait prendre conscience aux entreprises de la nécessité de se protéger efficacement contre les menaces internes et externes. Le principal moyen de contrer ce problème de sécurité consiste à utiliser des outils d'analyse du comportement des utilisateurs et des entités. Les spécialistes de la cybersécurité ont développé ces outils pour surveiller les activités des employés ou des utilisateurs des systèmes d'information des entreprises et détecter si leur comportement numérique post-connexion est anormal et nuisible. Ces outils étaient principalement basés sur des règles écrites par des experts, mais avec l'avènement de la science des données et des techniques d'apprentissage automatique, ils intègrent désormais des programmes plus intelligents et adaptatifs. Ces outils ont été développés comme une application de ces règles écrites par des experts et des techniques de science des données.

Ce type de logiciel assure le suivi des utilisateurs (c'est-à-dire les employés), des systèmes et du comportement des entités dans l'environnement de l'organisation. Ils utilisent des algorithmes pour apprendre ou modéliser

la nature du comportement numérique individuel, des techniques spécifiques telles que l'analyse statistique et l'apprentissage automatique pour différencier les comportements normaux des comportements anormaux. Les attaques d'insiders sont considérées comme un comportement anormal. Ce comportement peut avoir différentes origines, ce qui motive l'utilisation de multiples méthodes pour les détecter.

L'objectif principal de cette thèse est de proposer une solution à la question de la détection efficace des menaces d'insiders basée sur l'utilisation de la science des données et des techniques de calcul haute performance. Pour ce faire, il est nécessaire d'explorer différents aspects et applications de la science des données, de l'apprentissage automatique et des méthodes de calcul haute performance. Nous avons également utilisé le domaine de recherche de la modélisation et du profilage du comportement individuel des utilisateurs comme l'une des bases de notre solution. Outre l'environnement professionnel, ces techniques de modélisation du comportement individuel et de détection des anomalies sont utilisées dans des domaines tels que la détection des fraudes dans les réseaux sociaux, les télécommunications et la finance. Un parallèle peut être fait entre un insider et un fraudeur. Ceci nous a motivé à étudier le problème et les techniques de détection de fraude afin de trouver une meilleure solution contre les attaques d'insiders.

Le calcul haute performance jouant un rôle central dans l'analyse des big data, nous avons également étudié les méthodes, algorithmes et techniques permettant d'améliorer les performances de calcul. Nous avons utilisé une approche appelée unite and conquer, utilisée dans le domaine du calcul numérique, qui est essentiellement un modèle d'algorithme, de programmation et d'exécution adapté aux nouvelles architectures parallèles et distribuées.

L'objectif final de l'étude de ce domaine est de proposer de nouvelles approches de science des données et un logiciel hautement performant pour la détection des menaces internes. Ce travail visait à utiliser ce logiciel comme une extension des outils de gestion des identités et des accès, en fournissant des capacités de détection des menaces internes. Cela permet aux logiciels IAM de faire face à ce problème majeur de cybersécurité.

En résumé, les approches utilisées sont basées sur l'apprentissage d'ensemble, les méthodes basées sur les graphes, le PageRank, l'auto-encodeur avec des cellules de réseaux neuronaux récurrents, et les techniques de calcul haute performance. Nous obtenons des résultats satisfaisants en termes de scores de prédiction et de performances de calcul parallèle.

Acknowledgements

Contents

1 Introduction **10**

1.1 Area and focus of the PhD . . . 10

1.2 Motivation and context . . . 11

1.3 Problem definition . . . 14

1.4 Research questions to be answered . . . 16

1.5 Approach taken . . . 17

1.6 Contribution of the thesis . . . 17

1.7 Structure of the thesis . . . 18

I State of the art **21**

2 Insider threat state of the art **22**

2.1 Introduction . . . 22

2.2 Digital behavior and user profiles . . . 22

2.3 Insider threat type of attacks . . . 23

2.4 Industrial state of the art in the insider threat detection domain . . . 24

2.4.1 UEBA Gartner defintion . . . 24

2.4.2 UEBA compared SIEM . . . 25

2.5 Identity access management based UEBA . . . 25

2.5.1 IAM data for UEBA . . . 26

2.5.2 IAM based UEBA feature and use cases . . . 26

2.6 Advanced analysis methods for cyberattack detection . . . 28

2.7 Machine learning for insider threat detection review . . . 29

2.7.1 Unsupervised and semi-supervised learning methods . . . 30

2.7.2 Supervised learning and graph-based methods . . . 31

2.8 General discussion on the state of the art limits . . . 32

2.9 Conclusion . . . 34

3 Graph based method for fraud detection domain **35**
3.1 Introduction . . . 35
3.2 Subgraph analysis . . . 36
3.3 Propagation methods . . . 38
3.3.1 HITS . . . 38
3.3.2 PageRank . . . 39
3.3.3 Hits and PageRank for user behavior modeling . . . 41
3.4 Latent factor models . . . 42
3.5 Conclusion . . . 44

4 Unite & conquer and ensemble learning methods **45**
4.1 Introduction . . . 45
4.2 Related works to combination of the ensemble learning methods . . . 45
4.3 Imbalanced data . . . 47
4.4 Ensemble learning methods . . . 48
4.4.1 Bagging . . . 48
4.4.2 Boosting . . . 48
4.4.3 Boosting and bagging algorithms . . . 49
4.4.4 Bias variance trade-off . . . 50
4.5 Unite and conquer approach . . . 51
4.6 Conclusion . . . 53

II A new insider threat detection framework; from modelling to parallel implementation **55**

5 Modelling of the insider threat detection method **56**
5.1 Introduction . . . 56
5.2 Framework proposition basis . . . 57
5.3 Detailed contribution . . . 58
5.4 Proposed framework overview: UCEL . . . 59
5.5 UCEL mechanisms . . . 59
5.6 UCEL algorithm . . . 61
5.6.1 UCEL comparison with a mixture of experts . . . 63
5.7 UCEL co-methods . . . 63

5.8 UCEL for insider threat detection . . . 65
5.8.1 UCEL from data to behavior profiling . . . 66
5.8.2 User profile vs. Role profile . . . 67
5.9 Conclusion . . . 68

6 Integration of serial and parallel UCEL in an insider threat detection system 69
6.1 Introduction . . . 69
6.2 UCEL parallel programming paradigms . . . 69
6.3 Client-server parallel programming model . . . 70
6.4 Synchronized direct communication parallel programming model . . . 71
6.5 Multiple user handling programming models . . . 74
6.6 Insider threat decision making module . . . 74
6.6.1 Module abstraction . . . 74
6.6.2 Overall container-based architecture . . . 75
6.6.3 Employee risk scoring . . . 76
6.7 Conclusion . . . 77

III Framework validation by experimentation 79

7 Experiments context 80
7.1 Introduction . . . 80
7.2 Dataset of experiments . . . 80
7.3 Hardware architectures . . . 81
7.4 Performance metrics . . . 82
7.4.1 Machine learning methods performance metrics . . . 82
7.4.2 Parallel models performance metrics . . . 83
7.5 Software implementation tools . . . 83
7.6 Conclusion . . . 84

8 IAM based insider threat experiments 85
8.1 Introduction . . . 85
8.2 IAM audit data and experimentation description . . . 85
8.3 First experiment: user role clustering . . . 86
8.3.1 Clustering on the permission request data . . . 86
8.3.2 Outliers detection on the policy datasets . . . 87

8.4 Second experimentation: user request anomaly detection . . . 88
8.5 Third experimentation: supervised learning . . . 89
8.6 Graph-based IAM data visualization . . . 91
8.7 Conclusion . . . 92

9 CERT data exploration and preprocessing 95
9.1 Introduction . . . 95
9.2 Data exploration of CERT . . . 95
9.2.1 Data features . . . 95
9.2.2 CERT general analysis . . . 97
9.3 Prepossessing for the version R4.2 . . . 99
9.4 NLP based preprocessing for the version R6.2 . . . 100
9.4.1 Feature selection . . . 103
9.5 Activity frequency-based preprocessing . . . 107
9.6 Conclusion . . . 111

10 Classification experiments with UCEL on the CERT data 113
10.1 Introduction . . . 113
10.2 UCEL with anomaly detection co-methods . . . 113
10.2.1 UCEL compared to boosting . . . 114
10.2.2 UCEL and its co-methods evolution through cycles . . . 114
10.3 UCEL with supervised learning co-methods . . . 117
10.4 Scenario results . . . 118
10.5 Graph-based insider threat detection using PageRank . . . 119
10.5.1 PageRank based method class and UCEL test . . . 122
10.6 Autoencoder and LSTM based insider threat detection method . . . 123
10.6.1 Autoencoder LSTM class and test . . . 126
10.7 UCEL with new PageRank and Autoencoder methods . . . 126
10.8 Activity dataset test . . . 127
10.9 Oversampling techniques comparison . . . 128
10.10 UCEL applied to general classification problems . . . 129
10.10.1 Credit card fraud . . . 129
10.10.2 Seismic bumps . . . 130
10.10.3 Titanic passenger classification . . . 131
10.10.4 Fetal cardiotography classification . . . 132

10.10.5 Car state evaluation . . . 133
10.10.6 Accuracy comparison with original works . . . 133
10.11 Alarm system test . . . 134
10.12 Conclusion . . . 135

11 Parallel performance analysis 137
11.1 Introduction . . . 137
11.2 Conditions of parallel experimentation . . . 137
11.2.1 Metrics and hardware . . . 137
11.2.2 Libraries . . . 138
11.3 Parallel data management . . . 138
11.4 Synchronous control node model approach vs client-server algorithm . . . 139
11.4.1 First parallel models comparison . . . 139
11.4.2 Scalability test . . . 140
11.5 Execution time . . . 140
11.6 First asynchronous tests . . . 141
11.7 Handling multiple analysis of user . . . 143
11.7.1 PageRank and Autoencoder based UCEL . . . 144
11.7.2 UCEL with all co-methods and fewer ressources . . . 146
11.8 GPU impact test . . . 147
11.9 Conclusion . . . 149

12 Conclusion 151
12.1 Summary of thesis . . . 151
12.2 Some perspectives and future works . . . 152
12.3 Ethics reflection on data protection . . . 154

A First Appendix 155

B Second Appendix 157

List of Figures

1.1 IBM 2O16 cyber threat study [43] . . . 12

2.1 Cyber analytics tools funnel . . . 25

3.1 User behavior modeling models . . . 36
3.2 Example graph PageRank . . . 39
3.3 Adjacency matrix . . . 40
3.4 Transition matrix . . . 40

4.1 Bagging technique . . . 49
4.2 Boosting technique . . . 50
4.3 Unite and conquer approach for iterative methods . . . 52

5.1 UCEL framework model . . . 60

6.1 synchronous communication model . . . 72
6.2 Framework abstraction model . . . 75
6.3 Container based framework . . . 76
6.4 Risk scoring choices . . . 78

8.1 Mean shift on permission request dataset PCA 3D . . . 87
8.2 Mean shift on policy dataset PCA 3D . . . 88
8.3 Isolation forest on policy dataset . . . 89
8.4 Receiver operating characteristic curve for outliers detection methods . . . 90
8.5 User classification recall . . . 90
8.6 Graph modelization of a IAM permission request dataset . . . 92

9.1 Source of the data for the R4.2, scenario 2 . . . 97
9.2 Source of the data for the R6.2, scenario 2 . . . 98
9.3 Source of the data for the R6.2, scenario 3 . . . 98
9.4 Activity distribution in a training data sample, scenario 2 . . . 99

9.5 Activity distribution in function of the month for the R6.2, scenario 2 99
9.6 Activity distribution in function of the month for the R6.2, scenario 3 100
9.7 Pearson correlation for R4.2, scenario 2 101
9.8 Pearson correlation for R6.2, scenario 3 before NLP 102
9.9 R4.2-2 Activity distribution in a training data samples, scenario 2 103
9.10 R4.2-2 Activity distribution in a validation/test data sample, scenario 2 103
9.11 Suspicious word R6.2, scenario 2 104
9.12 Pearson correlation for R6.2, scenario 3 105
9.13 Feature selection table 106
9.18 Activity distribution in the test and validation data R6.2, scenario 3 106
9.14 R6.2-3 Features boxplot after NLP 107
9.15 R6.2-2 Activity distribution in a training data sample for the Scenario 2 107
9.16 R6.2-3 Activity distribution in a training data sample for the Scenario 2 108
9.17 Activity distribution in the test and validation R6.2, scenario 2 108
9.19 Activity data features histograms 109
9.20 Activity data time series 110
9.21 Specific Activity data time series 110
9.22 Activity data boxplot 111

10.1 MultipleRobcov(10) vs the best individual boosted Robcov 115
10.2 BP(4AD) vs individual boosted co-methods 115
10.3 MultipleRobcov(10) and co-methods evolution through cycles 116
10.4 BP(IForest, OcSVM, Robcov, LOF) and co-methods evolution through cycles 116
10.5 MultipleMLP(10) and co-methods evolution through cycles 117
10.6 PB(MLP, SVM, QDA, KNN, GNB) and co-methods evolution through cycles 118
10.7 KNN-Graph sample 121
10.8 PageRank generalized approach 122
10.9 PageRank distribution 123
10.10 K-Validation 124
10.11 MultiplePR(4) 125
10.12 Autoencoder LSTM Model 125
10.13 Reconstruction error distribution 126
10.14 Reconstruction error in function of the activity date 127
10.15 PB(4AD, PR, LSTM) 127

10.16 UCEL applied to the activity dataset, scenario 2 . . . 128
10.17 UCEL applied to the activity dataset, scenario 3 . . . 128
10.18 Oversampling techniques comparison . . . 129
10.19 UCEL for credit card fraud detection . . . 130
10.20 UCEL for seismic bumps classification . . . 130
10.21 UCEL Based on PR for seismic bumps classification . . . 131
10.22 Titanic passenger classification . . . 132
10.23 UCEL for fetal cardiotography . . . 132
10.24 UCEL for car evaluation . . . 133
10.25 Risk scorer comparison . . . 134

11.1 Parallel *behavior profilers* on GRID'5K . . . 139
11.2 Parallel *behavior profiler* strong scalability on GRID'5K (10 co-methods) . . . 140
11.3 Execution time of co-methods . . . 141
11.4 Asynchronous communication model . . . 142
11.5 Asynchronous vs synchronous model execution time test . . . 143
11.6 Asynchronous vs synchronous model speedup test . . . 143
11.7 Speedup of PageRank based method, 10 nodes for 10 users . . . 144
11.8 Execution time measure of PageRank based method, 10 nodes for 10 users . . . 144
11.9 Time measure of autoencoder based method, 10 nodes for 10 users . . . 145
11.10 Speedup of autoencoder based method, 10 nodes for 10 users . . . 145
11.11 UCEL speedup handling 5 users . . . 146
11.12 UCEL Time handling 5 users . . . 146
11.13 UCEL speedup handling 5 users . . . 147
11.14 UCEL execution time handling 5 users . . . 147
11.15 UCEL speedup handling 5 users . . . 148
11.16 UCEL execution time handling 5 users . . . 148
11.17 GPU Speedup . . . 149

List of Tables

2.1 UEBA use-cases out-of IAM scope 27

8.1 IGA dataset presentation 86
8.2 Anomaly detection methods metrics 88

9.1 CERT dataset presentation *R4.2* 96
9.2 CERT Dataset Presentation *R6.2* 96

10.1 Train and test with and without UCEL 119
10.2 Test results for 3 types of insider attack 119
10.3 Accuracy comparison UCEL vs original work 133

Chapter 1

Introduction

1.1 Area and focus of the PhD

Nowadays, the majority of businesses and corporations use digital solutions to manage their informational asset. These assets are a source of profit; hence, organizations understand the strategic importance of protecting their information systems. They use cybersecurity tools to shield themselves against all kinds of cyberattacks. However, these attacks continuously become a more significant issue because cyber-criminals understand the lucrative aspect of data and are continually and widen their attack panels.

For a long time, the cybersecurity specialists' main focus was the defense against external threats. They used tools based on databases of known threats against cyberattacks and monitored access to their internal networks, servers, and endpoints. Specifically, they used identity and access management (IAM) solutions to manage organizations' information assets' access. These tools provide and control the access of critical information and services for authorized individuals. They help to reduce the risk of loss, misuse, and sabotage of data, considerably. However, IAM solutions cannot identify users with legitimate data access that uses their access right maliciously. In the cybersecurity domain, this issue is defined as an insider threat. In a company environment, insiders are employees of the company who misuse their access rights. A typical example of insider activities from an individual is the sale of business information to competitor organizations in exchange for compensation (e.g., industrial espionage).

Another example is the sabotage of an organization's proprietary tools by an individual mandated by a rival company. According to IBM [43] in 2015 (see figure 1.1), 60% of cyber threats were due to insiders (i.e., 44.5% of malicious insiders and 15.5% inadvertent actors). In 2020, in the insider threat report [44] a survey conducted by the cybersecurity insider revealed that only 5% of the cybersecurity professionals interviewed feel invulnerable to insider threat.

The upsurge of this type of threat made companies realize that they need to protect themselves efficiently against internal and external threats. The main way to counter the security problem is to use user and entity behavior analy-

sis tools (UEBA). Cybersecurity specialists developed these tools to monitor the activities of employees or the users of business information systems and detect if their post-login digital behavior is anomalous and harmful. These tools were primarily based on expert-written rules, but with the advent of data science and machine learning techniques, they now integrate more intelligent and adaptive programs. UEBA tools were developed as an application of these expert-written rules and data science techniques.

This type of software keeps track of users (i.e., employees), systems, and entity behavior in the environment of the organization. They use algorithms to learn or model the nature of individual digital behavior, specific techniques such as statistical analysis, and machine learning to differentiate between normal with abnormal behaviors. Insiders attack are considered abnormal behavior. This behavior can have different origins, hence motivating the use of multiple methods to detect them.

The main objective of this thesis is to propose a solution to the issue of efficient insider threat detection based on the use of data science and high-performance computing techniques. To this end, it is necessary to explore different aspects and applications of data science, machine learning, and high-performance computing methods. We also used the research domain of user individual behavior modeling and profiling as one of the bases of our solution. Besides the business environment, these individual behavior modeling and anomaly detection techniques are used in domains such as fraud detection in social networks, telecoms, finance domains. A parallel can be made between an insider and a fraudster. This motivated us to study the fraud detection problem and techniques in order to find a better solution against insider attacks.

Since high-performance computing plays a central role in big data analysis, we also studied methods, algorithms, and techniques, allowing the improvement of computation performance. We used an approach called *unite and conquer (UC)*, used in the numerical calculation field, which is essentially an algorithm, programming, and execution model adapted to new parallel and distributed architectures.

The final purpose of the study of this domain is to propose new data science approaches and highly performant software for insider threat detection. This work aimed to use this software as an extension of identity and access management (IAM) tools, providing insider threat detection capabilities. This allows the IAM software to deal with this major issue of cybersecurity.

In summary, the used approaches are based on ensemble learning, graph-based methods, PageRank, auto-encoder with recurrent neural network cells, and high-performance computing techniques. We obtain satisfying results in terms of prediction scores and parallel computing performances.

1.2 Motivation and context

Cybersecurity is a domain that deals with the security of information technology (IT) systems. This is a highly strategic field due to the digital transformations in almost all areas. Indeed, cybersecurity is important in professional

Figure 1.1: IBM 2O16 cyber threat study [43]

and individual settings, respectively, with the digital transformation of numerous organizations and the popularization of connected communication devices such as smartphones.

One of the crucial stakes of cybersecurity is data protection. The development of new intellectual properties is one of the primary ways IT corporations make a profit. Their usual business model consists of establishing patents, the manufacture, and the sale of their products. They hence dispose of data resources linked to these sources of profit.

With the advent of data science methods, individual personal information exploitation also became a lucrative source of benefit. Computer systems can establish a profile of individuals that allows them to create customized advertising and proposed products and services to buy. Therefore, digital information related to individuals is also a source of lucrative information. These are two of the numerous ways that information assets can be used to make a profit. They highlight the economic importance of data protection. In healthcare organizations (e.g., hospitals) and companies that manage individuals' social information (e.g., Facebook, Twitter), there are many individual sensitive data. This information must be kept private to respect individuals' desire for confidentiality, and its access must be highly regulated.

Identity and access management (IAM), also referred to as logical access control (LAC), is a cybersecurity branch that deals with managing the individual's digital identity and access rights to information systems in an organization's environment. For example, IAM software can be used by corporations to manage the access of their lucrative data assets [42], or healthcare organizations to protect the personal information of their patients and their workers.

In general, data assets can have various forms regarding the activities of the companies or the organization. Businesses with digital systems can detain resources such as informational assets, which are confidential documents, industrial data, client databases, intellectual properties, proprietary applications, and services. Companies users of endpoints can have access to directories, applications, and APIs. In their internal network, companies place exchange directories to ease the sharing of files among employees. On web platforms, they put information that their employees can access to work out of their offices. These are data that enterprises want to keep their

control over. IAM software can manage access to all endpoints, internal networks, and Web platforms.

In order to vault this proprietary data, IAM systems control the identification, authentication, and authorization mechanisms for each organization member. They manage every individual that desires to access company data. These persons can be employees, contractors, or partners of the company in specific projects. They all are registered as a user of the IAM software to be certified as a member of their organizations and access their resources.

Identification For each IAM software user, the identification process provides means to be identified as a company employee or organization member. For instance, new employees will be assigned a new digital identity composed of a *username*, an *employee number*, a list of attributes (e.g., email address, smart card, certificates, Qr code), and the means to create a password.

Authentication The authentication process aims to reconcile user identification means with a password or another means of logging into the organization's digital systems. Usually, we used a password. This corresponds to a mono-factor authentication process. However, the authentication process can also be multi-factors. The multi-factors systems are commonly used in IT security environments because they provide extra layers of protection. It can be provided using smart cards or biometric sensors in addition to a classic password.

Authorization The authorization process manages the access rights to all company resources. These access rights provide information about the ownership of specific data and allow users to read, write or execute files and applications. The authorization policies are implemented using access control models such as role-based access control (RBAC), attribute-based access control (ABAC), and organization-based access control (ORBAC). The most commonly used models are respectively RBAC and ABAC. It also exists other paradigms, such as discretionary access control (DAC) and mandatory access control (MAC), which are respectively less secure [42] and less flexible than RBAC and ABAC.

Here we focus on RBAC since it is the most common solution in IAM software. RBAC is a paradigm of identity and access management that helps to manage employees' digital identity in the function of their roles. For instance, the role can correspond to the job of the employee in the company. Based on the principle of least privileges and the separation of duties, RBAC models are usually built upon an active directory system (ADS). In this model, at the identification phase, each user is affected to a group; group members have the same roles, and those roles are a set of permissions (i.e., access rights). It is possible for a user to belong to multiple groups. For instance, if they have various roles in the companies, or are working on a project involving numerous departments, or are managing distinct aspects of the development of a product.

To protect all their information systems against all cyberthreats, organizations usually combines IAM tools with other cyber-protection tools such as:

- *Antivirus and Firewall* to neutralize malware and programs that have the goal is to violate systems security systems
- *Intrusion detection* that help to detect intrusion of peers in the company network. They monitor the network and analyze the attributes of the packet transferred to detect anomalies.
- *Systems Information and Event Management* (SIEM) that are tools that help to monitor all the systems, endpoints, and network events. They mostly use correlation and complex event processing (CEP) to determine if an event is malicious.
- *Data loss prevention* (DLP) systems based on digital identity to allows the consultation of data. Their roles are to ensure that sensitive data is not shared outside the company
- *Endpoints detection responses* (EDR) tools that detect malware, virus, and zero-days attacks on endpoints
- *Cloud access security broker* (CASB) tools that has the goal to verify if cloud application respect the security policies of an enterprise. These tools can regroup all previously endpoint protection capabilities but for cloud architecture.

Using this combination of tools can be expensive due to licensing of numerous products. However, it guarantees almost a total protection of the companies assets against all external attacks.

Overall, IAM tools are very effective in protecting companies' assets, with strong authentication and authorization systems. It vaults the company's resources and protects them against unauthorized access. IAM combine with previously cited cybersecurity tools (e.g., IDS, SIEM, CASB) provides efficient protection against external threats. However, they do not provide any countermeasures against individuals who already detain access rights and who decide to behave maliciously (i.e., insiders). These are internal threats. Currently, they are one of the biggest concerns of companies working with digital systems.

1.3 Problem definition

Cyber experts are continually seeking new ways to counter the rapidly evolving landscape of cyber threats. Besides, hackers are continuously innovating and are adding tools to their arsenal. Cyber protection tools always have loopholes, and even if they are efficiently protecting digital systems today, they can quickly become obsolete. Nowadays, malicious insiders are the biggest concern for cybersecurity experts. Insider threat detection is one of the most troublesome issues in cybersecurity. This is due to the nature of insiders.

Against classic external threats, IDS can efficiently block potential intrusion in the organization networks [10]. They are mostly signature-based, using threat patterns databases, or anomaly detection-based using network profiling techniques. SIEMS can highlight anomalous systems events, and IAM software will restrict actions from any user without the correct access rights. However, against the insider threat, the task is more complicated because

criminals are either legitimate company employees or hackers who have stolen their credentials. Hence, they are not tampering with the network, and the events they generated are mostly assumed normal. This leverages the fact that specializes user behavior and entity analysis tools, and modules must be used to deal with this type of threat.

Using a simple definition, insiders are individuals who use their access rights to their organization's information systems with malicious intentions. These systems give access to information assets, which are critical resources that need to be protected. Since they already possess the ability to log on to the organization's systems, classic external threat detection tools, firewalls, and antivirus cannot stop them. Insiders can then endanger the company's assets without any means to spot them. There are mainly three categories of insiders: inadvertent actors, traitors, and masqueraders. We give here a quick definition of all the categories:

Inadvertent actors Whether it is conscious or unconscious, organization members' actions can endanger and negatively impact companies' profits. Most of the time, dangerous situations they provoke are the result of their negligence of security protocols. This category of insiders is called inadvertent actors. They are mostly unaware of security rules or lack knowledge about organization data use policies.

Traitors On the almost opposite, traitors are insiders that have legitimate access to company information systems and have malicious intentions regarding the usage of the organization's data. They have a level of knowledge of the weakness of the security systems, and light possesses the means to disable it or to give themselves privileged access rights (i.e., allowing them to bypass security protocols). Their actions are mainly motivated by the search for personal profit or the desire to follow a personal agenda.

Masqueraders Additionally to the previous type of insiders, hackers that can steal employee access rights using breaches and loopholes in the companies' security systems are also insiders. In this particular case, they are designed as masqueraders. They are black hat hackers that impersonated employees in the company system. They usually target high privileged users such as domain and system administrators. They can then use their security permissions to extract information or sabotage companies' assets. They know less about the companies systems than traitors but can do significant damage.

Depending on the companies' size, insider threat can cost hundreds of thousands of dollars to millions of losses [63]. This leverages the vital importance of finding solutions to these issues. Insiders usually target information resources such as databases, file servers, cloud applications, endpoints, networks, and customer information. Their objectives are diverse. The most common examples are the extraction and dissemination of confidential information, intellectual property, the sabotage of applications or data, the use of company resources for personal gain, and the use of company resources for personal gain.

1.4 Research questions to be answered

Nowadays, threats such as data leakage, fraud, industrial espionage, sabotage, and access abuses from privileged users are becoming more common. These are attacks perpetrated by insiders or fraudsters. User and entity behavior analysis (UEBA) software is used to stop insiders in the cybersecurity domain. For the fraudster, user behavior modeling (UBM) is the term used, but it follows the same principle of UEBA.

The companies use these tools to determine if employee behavior is normal or abnormal. Employee behavior is hard to classify because it can change over time. Its nature can diverge depending on their role, work requirements, and their company structures. These tools mainly use behavior modeling and anomaly detection techniques to detect insiders.

Most of the techniques used to identify malicious activities are based on the analysis of an activity dataset. This analysis is done with expert-written rules at a smaller scale and advanced data science methods such as machine learning methods [16]. When the studied dataset has a high number of dimensions, the rule-based methods tend to struggle because they cannot comprehend the nature of the data. In essence, since these rules a design by a human expert, they cannot deal with strange patterns that are identifiable with the analysis of a large number of variables. Besides the novel types and the slow pace, progressive attacks that are out of the scope of the defined rules are not detected.

In that case, data science and machine learning (ML) techniques such as supervised, semi-supervised, and unsupervised methods, and also graph analysis techniques are preferred instead. They also better manage the different scenarios of attacks with new training cycles since rules are specific to a unique scenario. ML methods can be used to build specific individual behavior profiles with the normal activity data and detect activity samples that diverge from this profile or find anomalous activities in the all-around company activity data.

The proposed UEBA solutions based on expert-written rules and ML are diverse [16, 37, 30], but they usually face the problem of high false-positive/negative (FP/FN), a lack of versatility to counter multiple scenarios of attack, maintainability features if they are only based on rules, and portability to a different business environment. Depending on the company targeted, a method initially performing well to detect an attack scenario can present an unstable detection accuracy. This is due to the diverse composition and properties of their activity dataset. It makes sense to optimize a specific model for a particular company, but the cost of the software development and maintenance of these tools can represent a drawback. The data volume is another big challenge to implement a detection model. Machines limited in their computation power struggle to treat the massive amount of data, and the creation time of the behavior profile can be impacted.

A solution to this issue would be to build a detection model somehow adaptive and able to manage efficiently different and extensive data input. This model would have to consider detection accuracy, detection time, portability, maintainability constraints and adapted to big data and high-performance architecture systems.

1.5 Approach taken

Advanced user and entity behavior analysis methods are based on the application of data science techniques on activity data. Depending on their size, a corporation can generate a huge amount of user activities and systems log data. This data volume can be challenging to handle by computers without highly performant hardware. This is an issue at multiple levels of a user behavior analysis problem. The performance of exploratory data analysis, data preprocessing, and model training, validation, and testing steps are impacted. In the case we deal with massive activities dataset, the elaboration of this insider threat detection solutions becomes a high dimensional data problem.

In recent years, there is a convergence of data science and high-performance computing (HPC) domain techniques to deal with a big data problem. HPC offers concepts to deal with high-dimensional problems. Techniques such as domain decomposition, communication management, and load balancing can be used to manage large datasets. There are also other ways to exploit the power of high performance architectures for high dimensional problems. However, there is a need to design customs algorithms to ensure the performance of the detection systems. However, there is a need to design customs algorithms to ensure the performance of the detection systems.

Since we are building insider detection software, we need to consider the extensible aptitudes of the chosen parallel programming models. This is an important step in building a detection system capable of handling the simultaneous monitoring of many employees. In addition, in terms of computational performance, the programming model must allow the different basic methods used to process a wide spectrum of data volumes. The learning time of the detection model is also an essential factor to consider. Since the targeted programming model can enable faster training of machine learning methods, it plays a significant role in getting the best time to protect business assets.

In this thesis, we studied and used several high performance computing techniques and proposed suitable parallel programming models to build a detection model with optimized computing performance. This optimization is largely obtained thanks to the characteristics such as the heterogeneity of the components, the asynchronous communications, the fault tolerance of the proposed global detection method.

1.6 Contribution of the thesis

In this work, we propose an insider threat detection solution based on the ensemble learning paradigm and a customized combination of bagging and boosting. This contribution is inspired by the previous work in the insider threat detection, fraud detection, behavior modeling domains, and the mitigation of high bias and high variance problems. The list below summarizes the contributions of the present thesis:

- Proposition of a new and efficient iterative machine learning framework, based on the application of unite and conquer approach on ensemble learning technique. This method can be seen as a particular combination of

boosting and bagging with a restarting strategy.

  - From the methodological point of view, this method offers means to accelerate the convergence of many classical ensemble learning algorithms and an algorithmic solution to remedy imbalanced dataset issues in applications such as insider threat detection.
  - From the point of view of calculation, the proposed method offers intrinsic heterogeneity, potential load balancing, fault tolerance, and the possibility of overlapping communications with computations, which makes it very well suited to today emerging high-performance architectures.

- A high-performance insider threat detection software framework, providing user and role-based behavior profiling based on supervised, anomaly detection, graph-based, and RNN machine learners;

  - A component library combining anomaly detection and supervised learning methods depending on the available data;
  - A component library for an insider threat detection method using graph analysis and the PageRank Algorithm;
  - A component library for insider threat detection techniques based on the use of an auto-encoder with RNN LSTM cells;
  - A portable model and its implementation of an insider threat detection module with corresponding alarm systems based on docker containers.

1.7 Structure of the thesis

In this thesis, we have reviewed various methods and algorithms in several fields such as cybersecurity, data science, linear algebra, and high performance computing. This in-depth review has helped us create an effective solution for analyzing user and entity behavior to detect insider threats, which is one of the main issues in cybersecurity.

More precisely, we propose an insider threat detection method based on a combination of an approach used in linear algebra, called unite and conquer, and ensemble learning methods to build an insider threat detection framework. This framework uses the principle of unite and conquer, consisting of making collaborate several machine learning methods to build individual post-login user activity profiles and accelerate the convergence of the training of the classifiers. These profiles are used to identify a new activity as normal or abnormal. The base methods of the ensemble of learners belong to the family of the unsupervised/supervised machine learning and graph-based methods. We show that this approach makes it possible to obtain significant gains in accuracy relative to the individual

base methods. We also implement it with high-performance parallel computing techniques and show its efficiency relative to scalability and in terms of execution time.

This framework is integrated in a global insider threat detection module based on docker architectures. This system is designed following the ensemble learning principles and takes into account high-performance computing, cloud architectures, big data, and its usability as an IAM software extension. These architectural choices ensure its portability to multiple on-premise and cloud environments and its ease of combination with other cybersecurity tools. This system furnishes an insider attack risk score in the function of the prediction of the different detection methods.

This manuscript is divided into three parts representing respectively the states of the art, the conceptual basis of the contribution, and the experimentation phase.

In the first part, chapters 2, 3, and 4 present the state of the art of insider threat detection, some graph-based methods for fraud detection, and the conceptual basis of our contribution. Chapters 5 and 6 constitute the second part. The modeling of the proposed insider threat detection solutions is presented in chapter 5. Chapter 6 proposes the parallel programming models and some implementation details for the proposed techniques.

In the third part, chapters 7 to 11 present the details of our experiments. They allow to validate the proposed methods and framework as well as our insider threat detection system. We describe the conditions of these experiments and present the results and their analysis. Finally, Chapter 12 concludes this work, presents some perspectives and discusses the ethical concerns related to these questions.

Part I

State of the art

Chapter 2

Insider threat state of the art

2.1 Introduction

To detect insider threat attacks, cybersecurity experts use user and behavior analysis software. These tools monitor user activities and determine if they are not harmful to their organization. UEBA software is based on behavior modeling techniques. Specifically, the UEBA software uses a combination of techniques using expert-written rules, data science techniques, and graph feature analysis [16]. At the beginning of this work, most of the existing solutions were based on experts written rules. In this section, we study the state of art of the insider threat detection domain and user behavior analysis. These domains are linked to the cybersecurity domain but also behavior analysis and fraud detection. We started by defining some context and presenting some tools used in the cybersecurity industry to stop insiders. We then study the cybersecurity technologies using behavior analysis and particularly focus on research specifics to insider threat detection. This helps us to propose an original UEBA solution that handles some issues such as numerous false alarms that detection methods face today.

2.2 Digital behavior and user profiles

Digital behavior and user profiles are one of the central concepts linked to user behavior analysis. In their review of user profiling in intrusion detection systems, Peng et al. [61] present a definition of the concepts of user behavior, system behavior, and the expected behavior of a profile. A user is an employee or any individual using an organization authentication system to access the information systems. An entity can be defined as digital systems or hardware such as internet of things (IoT) objects that generate data during their period of operation.

A digital behavior can be seen as a data collection of the user's interaction to information systems of their companies after their login. It is essentially the data generated when users directly interact with the information systems. For instance, the behavior of a user can be the ensemble of its post-login activities. A behavior profile is

then a subset of the activity data with only common or normal behavior for a unique individual.

Systems or entity behavior is the data collection of systems interactions with other information systems of the company. It is data generated by the hosts and the network systems. They can be indirectly due to users actions. For instance, we can't talk about systems behavior when users are active on a host software, and activity data is generated by the internal component such as the CPU, the memory, or other background programs [61]. Another example would be the sensors that data that are generated by some IOT objects when their make measures on their environment. A user profile can be seen as a model of the usual behavior of the user. The same analogies can be made for systems with a systems profile.

The samples of a profile can be seen as belonging to the same normal of activity. This consideration helps to turn the insider threat detection problem into a classification problem. With this consideration, to detect insider threat, the goal would be to classify a new event as being the same member of the same class of the profile. However, if there is an anomalous element in the original activity record, it is maybe preferable to use an anomaly detection approach. However, for the anomaly detection approach, it is necessary the normal class is dominant over the anomalous class. It is important to note that an anomaly that does not necessarily mean an insider attack. Further investigation is always needed to characterize the nature of an anomaly.

2.3 Insider threat type of attacks

In an industrial sense, there are typically five types of insider attack: account comprises (highjacking and sharing), high privilege access abuse, data exfiltration, cyber fraud. Account compromise is a typical mascarade attack. Users have their credentials stolen by hackers, and they take profit from the situation. They can ask for ransom for the companies data or share their secret with the public. High privileged access abuses are attacked from the individual with administrator status that has the possibility to allows access right. In a sense, they have the key to the kingdom, and they can do malicious action without being discovered if they are not monitored. Data exfiltration can be perpetrated by any user who has access to sensitive data. The data transaction must be monitored closely in order to avoid critical information being shared outside the companies.

Cyber fraud is specific to companies that deal with accounting, treasury, and fund transactions. Their information systems deal manages critical data. The users of these companies followed strict guidelines. However, there are still occurrences of fraud in this type of company. UEBA solutions are hence essential to have an extra layer of protection.

Within this possible type of attack, multiple scenarios can occur. In this thesis, we specifically deal with a list of insider threat scenarios that are close to the IAM context. These scenarios are from the computer emergency response team (CERT) of the University of Carnegie Melon Work [35]. This team is one of the main research laboratories that develop a solution for the insider threat detection domain.

2.4 Industrial state of the art in the insider threat detection domain

2.4.1 UEBA Gartner defintion

For years cybersecurity software makers were focused on defending IT systems against external threats (i.e., antivirus, firewall, intrusion detection systems). Today companies with digital systems are more likely to be put at risk by insider threats than by external threats. As solutions to this problem, many detection tools are based on the establishment of the verification of the compliance of users to security rules written by cyber experts. Violation of these rules leads to the launch of alarm systems or contextual decision-making. There are also tools to analyze the correlation of systems and user events. In this context, events are user or systems actions at a specific time. With the advent of data science, solutions based on more intelligent algorithms are emerging.

From an industrial point of view, cybersecurity specialists have developed tools to analyze the behavior of individuals to counter insider threats. They can be referred as user and entity behavior analysis software (UEBA) or user activity monitoring software (UAM). Our study of the industry has found software companies such as *Gurucul*, *Splunk*, *IBM*, *Exabeam*, *Fortscale*, and *Securonix* that offer solutions based on advanced analytics methods in other words data science. User and Entity Behavior Analysis (UEBA) is a term coined by Gartner in 2015 [16]. It refers to the use of advanced analytical methods to effectively detect insider threats. Gartner specifies that these tools can detect deviant behavior, detecting access abuse, identify compromised employee accounts and terminals, and detect new types of threats. These are synonyms to the type of insider attacks presented earlier. UEBA solutions also propose features such as data collection from multiple security silos. They are most often in the form of stand-alone software or are modules integrated with SIEM (Security information and event management) software.

In general, UEBA software have three standard building blocks to deal with insider threats. The blocks manage the collection of information and data on employee activities, the employee behavior modeling systems, and the alerting system. The part corresponding to the collection and record employees actions uses software wrappers. These wrappers are software tools that collect all data and indicators of activity on corporate networks and endpoints. The behavior modeling part of UEBA systems builds behavior models of employees that can determine if new unlabeled events are normal, abnormal, common, or uncommon. In other words, these profiles are used to analyze the new event and predict if they are a danger to the business. Behavior modeling is the heart of this system. This can be based on a comparison to individual past actions or his peers behavior. The alert system is a system that establishes in real-time a risk score or threat level corresponding to employee behavior. This allows you to trigger specific alerts or apply contextual actions if the risk score reaches a chosen threshold. The risk score can be shared with other cybersecurity tools.

2.4.2 UEBA compared SIEM

If we look closer look at SIEM, it is noted that the main difference between these systems and UEBA solutions is the use of advanced analysis techniques. These techniques can be defined as using data science methods or more sophisticated methods. Classic SIEMS uses correlation study and complex event processing (CEP) based on expert-written rules. The use of rules and correlation studies is not efficient with extensive data with numerous dimensions since they need a lot of human intervention and expertise. Data science methods, specifically machine learning techniques and deep learning, are built to deal with massive data that humans do not easily manage.

As the types of cyberattacks change, threat databases and correlation rules need to be updated regularly. These processes have a high maintenance cost. Therefore SIEM suppliers are starting to integrate UEBA modules into their products. These two solutions share some common characteristics. First, they use huge amounts of data collected by users and entities and apply algorithms that analyze events and generates alerts. Some UEBA tools use SIEM systems as data providers. SIEM tools use UEBA solutions as a behavior analysis module. A study by Gartner [16] predicts a merger of the market for these products by 2020.

Other products such as CASB also add UEBA capabilities or use machine learning techniques such as EDR to detect the type of threat they are exposed to (figure 2.1). In a sense, a UEBA module can be integrated into every cybersecurity tool for users and entity activity analysis or to add an information input.

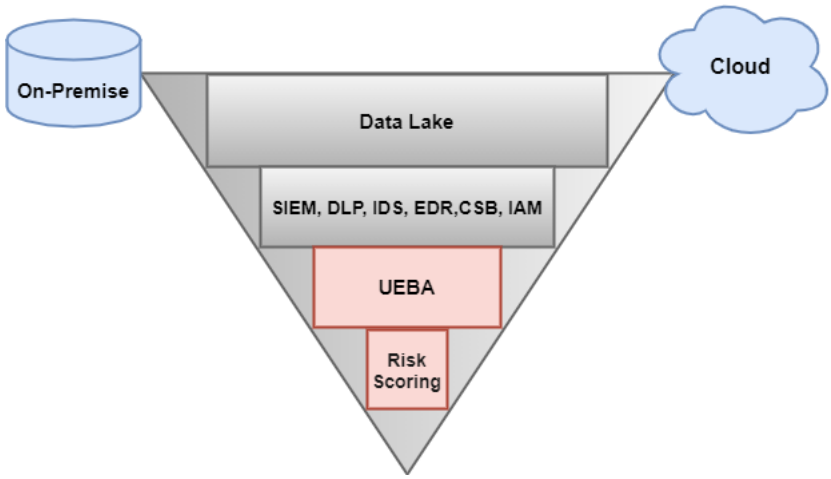

Figure 2.1: Cyber analytics tools funnel

2.5 Identity access management based UEBA

We define in the section 1.2 IAM tools as software used by companies to manage the digital identities of workers and access to their lucrative data assets [42]. These tools can be used in many fields, such as healthcare organizations, to protect the personal information of their patients and workers.

2.5.1 IAM data for UEBA

Specifically, using only IAM data based on access rights, authorization, and authentication, it might be complicated to build an insider threat detection system. This is deeply linked to the quantity and type of data the IAM software provider dispose of. Typically, IAM tools present mainly three modules. These are modules that deal with identity and governance administration (IGA), web access management (WAM), en endpoint access management (EAM). These tools only dispose of information on the list of authorization of a user, the events of the right attribution, and the chronologies of access to authorized applications. IGA manages the permission and access rights attributions to users. WAM and EAM control, respectively, the authentication and the connection to third-party light web-based applications or heavy clients software.

Data is critical to building UEBA capabilities. The information provided by IAM tools is generally limited to authentication, the listing of the authorization, and the record of the timestamps of access to the application context. However, there are detectable scenarios such as account compromise by session tracking (ACST) and high privileged access abuse (HPAA). In order to counter insider in a larger sense, behavior during the inside the used application is useful (e.g., Website visit in browser, mail recipient in the electronic message client). In other words, the logs of applications accessed by users could be useful to UEBA software.

2.5.2 IAM based UEBA feature and use cases

There is a list of specific actions related to a UEBA we can do if we only dispose of IAM data. Identity governance, web access management, and endpoint access platform data can help to detect some of the insider threat attack scenarios presented before.

Compromise account and session tracking For this case it possible to track users that log on to a web or on-premise platform. Firstly, it is necessary to recover the data of users interaction to WAM and EMA systems when they use these platforms to access applications. These data can be clustered following the principles of sessions which are, the ensemble of events in a working day or between the first login the last logout event of the day. A session can be seen as a data subset of daily interaction between a user and the WAM or EMA software. They can help study the authentication and connection to applications habits of the users and its usual working hours to establish is in a behavior profile. This profile can then be used to detect if new activities are in the session habit of the user or are anomalies. They can also be used to verify if there is an abnormal number of activities and request connection and authentication in a short time frame. Finally, they can check if there is a login from the same user at the same time with different IP addresses. This could be indicative of the masquerade type of attacks. For this purpose, machine learning algorithms such as anomaly detection, novelty detection, clustering techniques, and graph feature analysis methods can be used to detect insiders. We will discuss details about these methods in

future sections.

High privileged access abuse In this case, it is possible to use IGA, WAM, and EMA data to identify a user with a high privilege account (HPA) that behaves maliciously. The goal here is to check if the HPA is attributing too many critical access rights to a user. If this attribution is followed by suspicious activities or transactions, then the attack is confirmed. The detection of this type of attack requires monitoring permissions attribution and tracking the user session (i.e., CAST). Using IGA data, it is also possible to check if a user does not have too many rights in the function of their roles and compare them to other users with the same roles. These actions can help detect outliers in the organization. However, they are not necessarily insiders. Here, also, we can use data science methods such as anomaly detection method, clustering technique, and supervised classification.

Adaptive authentication UEBA systems allow building contextual decision systems such as Adaptive authentication (AA). This is an advantage of using a risk score or threat level from the ACST and HPAA cases. We can use this score to recommend a level of authentication. When the risk score is low, the user will need one factor, but if the score is high, two or more authentication factors. This allows protecting information assets with prior knowledge on the user behavior. We give more detail about the risk scoring in the modelization and validation chapter.

Self-Audit and ID Theft From IAM data, it is also possible to build a self-audit and ID theft (SAID) feature. This can be done by building a system that detects unusual login or authentication events outside of the user session behavior(e.g., late-night, the weekend, or its vacation days). The self-audit feature will then send directly to the user a report with this activity and ask to confirm it. He will have the role to prove that it was his activity, or it will be classified as identity theft.

Other UEBA attack scenarios exist. However, they need more data than IAM solutions can provide. We give some examples and the types of data they need in table 2.1.

Table 2.1: UEBA use-cases out-of IAM scope

UEBA Use-case	Means
Insider thread Deterrence	Use real world insider threat incident dataset to train UEBA model and use them for real companies
Cyber Treat Deterrence and Trusted Host and Entity compromise	Using data from other cybersecurity tools such as application logs, SIEM,EDR, CASB to model systems and user behavior, UEBA modules are able to detect a larger panel of attacks
Cyber Fraud Detection	With treasury, accounting and payment datasets it's possible to detect fraud using the same approach that UEBA use to detect abnormal behavior

2.6 Advanced analysis methods for cyberattack detection

Early work on behavior modeling was used in intrusion detection systems (IDS). In this case, these tools are also known under the name of network behavior analysis (NBA) software. IDS vendors were the first to use machine learning to detect abnormal behavior on organization networks. Reviews that are presenting behavior analysis methods for IDS tools are numerous (e.g., [10], [61]). They mostly focus on detecting anomalies at the network level but also present insightful definitions of behavior and profiling.

According to Bukzak et al. [10] survey for data science and machine learning techniques for IDS, there are two types of solutions to detect cyber attacks. These solutions are based on behavior profiles constructed using supervised machine learning methods that require labeled data or unsupervised machine learning methods based on clustering algorithms.

Specifically, to detect these cyber-attacks, (IDS) uses sensors, build network traffic profiles, and track any changes in the usual data flow of network packets. Sensors are used to monitor several attributes of transferred packets, such as their sources, destinations, or size. They can thus detect attacks that target companies networks, such as denial of service (DS), distributed denial of service (DDS), and port scan (PS). The use of profile in IDS inspires some part of our contribution. A profiling approach is not necessarily used to detect insider threat detection.

In [11], Casas et al. presented a network intrusion detection system based on unsupervised methods (UNIDS). Their approach detects cyberattacks without using labeled data and offers a model that does not need learning time. They combine three algorithms to detect intrusions. The first is a method that detects a change in the data flow of network packages. This algorithm is based on the time-series study. They then applied density-based clustering and subspaces clustering to group data flows and spot outliers that do not belong to a particular cluster. These outliers are seen as indications of an attack on the network. However, since they used multiple methods, their algorithm is slow and present a high rate of false positives. These are common issues on user behavior modeling to detect cyberattacks. This article inspired parts of our contribution with the use of methods based on time series analysis.

Another example of behavior detection can be found on workflow-driven web applications. Xue et al. [52] used hidden Markov models to create a tool capable of countering attacks on web applications. These attacks can be the falsification of intersite queries and workflow guidelines violations. Workflows contain tasks that use data, which follow a specific execution order. Hence, for a web application to work correctly, the order of use of the data is essential. The authors use a hidden Markov model to predict the next state in workflows (i.e., the next stage of executions). Comparing the actual state and the prediction, the model determines whether certain processes in these workflows are abnormal.

Using the combination of access control-based tools with a genetic algorithm, Bradford et al. [42] proposed a method to detect insider threats. They based their research on work on the use of genetic algorithms in cybersecurity. They call their approach the small user world principle. Their assumption is that users in the same role perform

similar actions on or with company resources. Essentially, they mapped the roles of the users to be processed using genetic algorithms. They then investigated the match and mismatch of user actions based on their access to the permissions of the associated roles. This article leverage the usefulness of individual comparison to peers to detect unusual activities.

Concerning the detection of masquerade attacks, work has been carried out on the *Schonlau* dataset. This set consists of the keyboard commands of the users in the operating systems of the company. The algorithms used to detect malicious activity are inspired by text classification methods based on the Bayesian statistical classification method. These techniques determine the subject or general themes of the texts, using the words that compose them. Schonlau and his colleagues used six masquerade detection methods to identify compromised employees, namely [67, 68, 56]: Bayes 1-step Markov, hybrid multi-step Markov, incremental probabilistic action modeling, uniqueness, sequence match, and compression. This work leverage the fact of using text classification techniques to detect insiders. In our contribution, we used natural language on text features to get more behavior information.

There is also a way to detect abnormal activity through the use of decoy documents. These documents are bogus files that do not disclose important information. They are meant to trap insiders who try to gain access to valuable information. This insider detection system is based on research on honeypots. In their work, Bowen et al. [6] have proposed a decoy document generator, an access detection sensor, to detect insiders. Decoys are made to launch alerts when an individual accesses them. The approach taken by Bowen and his colleagues is to create decoys based on custom tagging systems. They see traditional user anomaly detection tools as having problems due to the unstable nature of the behavior. They also presented their perspective on the difficulties encountered in determining insiders skill levels. They could potentially disable certain security systems and thus fall under the cracks. This is specific to the issue of HPAA that we can deal with using IAM data. In this thesis, we do not specifically use decoy documents, but we plan to do it in futures work to improve the arsenal of our framework. The use of decoy documents can be considered as an advanced analysis technique out of the scope of data science.

2.7 Machine learning for insider threat detection review

According to Gartner, [16], existing insider threat detection tools already rely on different detection models. For each case, the input data set can be different. Hence, the most efficient algorithm depends on the input data. To detect insiders, most of the software uses a combination of techniques such as expert-written rules and data science techniques [16]. Besides the previously presented techniques, methods based on machine learning are used specifically to detect insider threats. These techniques are usually in the spectrum of supervised, unsupervised machine learning and graph-based methods. This large panel of approaches is the consequence of the heterogeneous nature of behavior. The representative data can have different structures, and there are multiple ways to model and classify behavior. Even though labeled data is harder to come by in this domain [61], the supervised method of-

ten outperforms the unsupervised learning and the graph-based algorithms. The domain researchers usually test their methods using artificial data. Also, some works combining these different approaches exist. They showcased promising results to fix this cybersecurity issue. We focus on insider threat detection methods based on machine learning and graph analysis in the following subsections.

2.7.1 Unsupervised and semi-supervised learning methods

At the academic level, a lot of research has been done in the field of insider threat detection using unsupervised learning. Most of the work is centered around the use of unsupervised anomaly or novelty detection tools. However, some of them used semi-supervised techniques. Due to the nature of insider attacks, using anomaly detection seems to be a reasonable decision to handle this type of issue. Normal employee behavior is often the dominant activity class, and the insider action is the minority in the data. Anomaly detection methods are mostly unsupervised classification methods that deal with an unbalanced dataset. Semi-supervised methods are classification methods that use a small portion of labels from the dataset.

Sun et al. [72] proposed a comprehensive unsupervised method of internal threat detection based on the isolation forest (IForest) method. An isolated tree forest is an anomaly detection algorithm using several binary research trees and that classify data samples by observing the depth of the branches. Short branches compared to average depth are considered abnormal. Trees are built individually, choosing branch separation properties randomly. Sun et al. tested their solution with a dataset of company staff accessing the payroll tool. Their algorithm extracts a set of user characteristics and creates their profile based on a collection of extended IForest methods. They present this extended method as a modified IForest to support categorical data. The problem that can be encountered by using an isolation forest algorithm is that the training data must be mostly healthy. This problem can be more or less remedied by putting several trees working with samples of data (i.e., Bagging) or by carrying out several iterations of the algorithm. However, this solution remains expensive in terms of computing power.

Tuor et al. [76] proposed unsupervised online approaches based on a deep neural network (DNN) and a recurrent neural network (RNN) (i.e., using an LSTM architecture) as a prospective filter for a human data analyst. Their goal is to diminish the workload of a security analyst. Their best model obtained an anomaly score in the 95.53 percentile for the insider activity. This online approach shows great results. However, as this is an unsupervised approach, human intervention is needed to determine the causes of the anomalies. This work inspired a part of our contributions centered around the use of artificial neural networks for insider threat detection.

Haidar et al. [39] gave insights on how to boost two anomaly detection methods. They proposed an overall semi-supervised method, using two methods: a one-class support vector machine (OcSVM) and an IForest. They also proposed a progressive update method, using false positive (FP) packages identify by a domain expert. They are using this new information to improve their previously generated model. Using their basic classification methods

on the initial data, they apply a class decomposition technique using the K-Means algorithm on normal labeled data to obtain k-clusters. On these subsets, they reapply their basic algorithm to detect anomalies at a more local level. The progressive update system optimizes the algorithm by oversampling results classified as FP. The purpose of this is to create a more precise decision limit for basic algorithms. The decision limit is a logical separator between normal data and outliers establish by classification methods. The decision limit can be in the form of a straight line (2D) or a plane (3D), depending on the dimensions of the dataset. In this work, Haidar et al. handle behavior model exception, but not its possible evolution over time. This can be handled by a periodic training scheme combined with an intelligent windowing of the training data. This algorithm presents an original way of dealing with the insider threat problem. However, it requires the intervention of a human operator for its optimal functioning. We used some insight from this work to develop our contribution.

In conclusion, unsupervised or semi-supervised approaches are the most suitable without access to a lot of labeled data. However, these approaches cannot characterize the source of the problem, which means that they can spot unusual or uncommon data samples but cannot give insight into their causes. This implies human intervention which is always needed to determine the grounds of the anomalies.

2.7.2 Supervised learning and graph-based methods

In the overall literature, supervised learning approaches showcase better performance than unsupervised methods(e.g., anomaly detection). This is particularly true for approaches based on neural networks. Yuan et al. [80] presented a deep learning approach to detect insiders. They use a combination of long short term memory (LSTM) and a convolutional neural network (CNN) to identify the abnormal behavior in multiple scenarios. They obtained an area under the curve (AUC) of 0.94 for their classifier. However, to work properly, they need substantial and balanced data. Since the activity dataset naturally contains more normal activity samples than insiders, this method can struggle to perform well and be susceptible to bias and variance issues.

Graph-based methods are another approach to tackle the problem of insiders. Based on the graph features analysis, the detection mechanism is to spot anomalous nodes or subgraphs. They share the same issue as previously discussed, anomaly detection approaches. They are also sensitive to exceptions and do not give information on the nature of the anomaly. In the domain of insider threat detection, they are combined numerous times with a supervised or semi-supervised approach.

Chen et al. [13] used bipartite graphs to map user access logs. They then use the *cosine similarity* method to get the resemblance between employees and to detect if specific access is malicious or not. They based their approach on a correlation method used in collaborative filtering techniques for recommendation systems. This system depends on the amount of information used to determine the model and to have reliable prediction results. This is a common problem with recommendation systems. It might be interesting to use a decomposition in singular

value as a remedy.

Moriano et al. [57] also use bipartite graphs to detect internal threats. Somehow, they study changes in users behavior over time to detect deviant behavior. They focus specifically on what they call precipitating events and use them to detect abnormal behavior. They define these events as key events that could indicate that an employee is becoming a threat to the company. They applied their approach to a dataset of a version control tool for software development. They model normal behavior using bipartisan graphs representing user interactions on software components and thus compare the volume of interactions before and after precipitating events. Upon discovering a significant increase at this level, they designate these precipitating events as signs of internal threat. This approach encounters detection issues if there is a gradual and non-sudden change in an employee's behavior with malicious intentions. This model would then be unable to detect insiders since there is no abrupt change.

Gamachchi et al. [33] proposed using a graphical processing unit to extract the properties of the multidimensional data graph representation of the CERT at Carnegie-Mellon University. These are more specifically induced subgraph properties that characterize the global graph at a local level. The characteristics of the graphs are then used to power an IForest algorithm to detect deviant behaviors. In this work, the studied behavioral parameters are chosen by an expert human operator of the field. This method may be effective but does not take into account parameters that could be internal threat indicators that cybersecurity experts would not be aware of.

Parveen et al. [59] also proposed an overall method combining supervised, unsupervised learning methods and anomaly detection based on a graph-based study. Their graph method is based on the use of the minimum description length (MDL). By comparing three method families in terms of the accuracy of the results, they proved that an OcSVM outperforms conventional support vector machines and the MDL-based graph method. Previously, they had proposed an overall method based solely on the combination of several one-class support vector machines in addition to a flow extraction approach to take into account an individual's time-changing behavior. Starting with an overall method, Parveen et al. decide to use a conventional voting method. However, by adopting this kind of approach, if the accuracy of some classifiers is very low compared to others, they may represent a disturbance in the quality of the results. In their case, the performance of their supervised algorithm is far superior to those of their unsupervised method and those of their graph-based method. This shows that in the case of an ensemble approach, the choice of base methods is important. A combination of this type does not allow to take advantage of all base methods. It is more interesting to combine methods within the same family of the algorithm (i.e., a combination anomaly detection method).

2.8 General discussion on the state of the art limits

The most used insider threat detection software are based mainly on expert-written rules. However, they are using progressively advanced techniques of data science (e.g., machine learning methods, graph-based approach, etc.).

We pointed out that much of the research on user behavior modeling deals with IDS [61, 10].

Expert-written rules have limited capacities to detect insiders. This is due to the changing nature of the behavior according to the time and function of the studied environment. Also, they are very limited by the dimension of the studied activity data. Rules are based on the value occurrence of a variable of the data, which makes it suitable for categorical data, however not when there are too many categories. If the data dimension is too high, the rules become too complex to establish and comprehend.

Hence, data science techniques and machine learning approaches are more suitable with higher dimension behavior data. The more ancient research based on data science has focused more on detecting masquerade-type attacks than other types of insider threats [67, 68]. The research effort was then switched to anomaly detection methods. However, the use of this approach is known to cause a large number of false alarms. The number of false alarms is a consequence of a high rate of false positives. Non-detection (i.e., false negative) also issues since insider activities pass as normal.

Overall, we notice an increased rate of misclassified data (i.e., FP/FN), as an incidence, on the number of occurrences of false alarms or non-detection of attacks. The reduction of the FP/FN rate is a key asset for UEBA solutions. One of the causes of this high rate of FP/FN is the lack of real and relevant data, preferably labeled, for insider threat detection. Such data are essential to establish reliable behavior detection methods. Databases used to work on insider threat problems are very rare. The early proposed machine learning solutions are tested on artificial or proprietary databases that only respond to well-defined scenario cases.

The source of this issue is the quality of the data. The companies do not necessarily have systems to store their employees post-login activities and are often reluctant to share their business data. Their employees may also be opposed to the consultation of their private information and about their activities during their working hours. Hence, it is practically impossible to get user activities dataset labeled for specific companies.

Regarding these UEBA tools lack of versatility, the nature and availability of data differ from one company to another. A detection method can be optimized for a work environment, but it will not be the best option for a company. It may even be ineffective when there is a change in the attack patterns. The training time of machine learning methods depends heavily on the size of the input data. Performance in terms of detection time is correlated to the amount of activity data. This constitutes an essential factor to consider when designing a solution.

This implies that solutions based solely on supervised methods trained with artificial data, even if they are presenting the best accuracy, are not universal and may be more or less irrelevant to real study cases. Besides, when this data is not sufficiently diverse, classifiers are likely to experience overfitting problems. Conversely, when the data samples are not majority following the same model, the classifier is likely to face underfitting issues. Overfitting and underfitting issues also increase to have the risk of having FP/FN.

Finally, in the academic literature, examples of applications detecting insider threats in the specific context of access and identity management are very rare [42, 72]. Data from the main process of IAM is useful to detect some

insider threat use cases such as CAST, HPAA, SAID, and AA. However, they can not detect data loss, cyber fraud, and other types of internal attacks. Insider threat detection is treated at a larger scope in the research domain.

2.9 Conclusion

The literature associated with insider threat detection is rich in different types of detection approaches. Compared to methods based on rules and statistical correlation, data science-based methods present less adaptability and maintainability issues. For example, if we change the organization environment, rules need to be re-written where are well contrast machine learning pipeline just needs new data. Nevertheless, the data science base approach suffers from a high rate of false positives (FP) and negatives (FN) in their detection phase and lacks versatility if they are not in a stand-alone manner. In the research domain, they are trained with rarely real business labeled data or synthetic scenario-based data. As a consequence, these solutions are not able to counteract multiple scenarios or new types of attack efficiently.

Since the begging of our work, we have seen a shift in research efforts on the topic of graph analysis and ensemble methods to detect insiders. It exists techniques that combine both of these methods. The major problem with detection tools is the rate of misclassification. One of the reasons for this high FP/FN rate is the overfitting or underfitting issues for classification methods. There is also a lack of consideration of the computation performance of the detection methods.

This thesis is a contribution to remedying these problems for insider threat detection. To reach this goal, we first studied research works dealing with data science by focusing on graph-based user behavior modeling and ensemble learning approaches for insider threat detection problems. We also studied the literature on the combinations between several unsupervised and anomaly detection methods, the combinations between graph method and machine learning methods, and the development of methods based on deep learning combined with a graph-based approach.

To build our detection system, we study these approaches in the next chapters. We follow similar directions for our contribution. However, we focus on prominent issues such as the rate of false alarms, versatility, maintainability, and computation efficient constraint.

Chapter 3

Graph based method for fraud detection domain

3.1 Introduction

In some IT domains such as social networks, web page distribution, and recommendation sites, fraudulent user activities are an issue that cybersecurity experts are continuously facing. Similar to some aspects of insider threat detection, this issue leverages the use of behavior analysis. In these fields, the term ”user behavior modeling” refers to the study of digital users' behaviors in order to classify them as normal or fraudulent. The fraudulent actions are behavior anomalies, same as insiders' activities. With some considerations, the fraudsters can be considered as the equivalent to insiders. The methods proposed to deal with this issue use graph feature analysis, data science, and linear algebra. We can represent the activity data of these domains using nodes and edges. Graph features are then critical information to detect anomalies.

For example, in the social network domain, a node can represent a user page. An edge between two nodes would then correspond to their friendship or following relation. We can also represent graphs under a sparse matrix format. This allows using matrices and sub-matrices characteristics to draw conclusions from the original data.

Datasets are usually substantial in these domains. A combination of linear algebra and scalable, high-performance computing methods represents an efficient way to compute graph properties.

Overall, in the user behavior modeling literature, there are three main families of techniques. They are namely the subgraph analysis methods, the propagation methods, and the latent factor models. In the following section, we present several works and applications for each one of these techniques (figure 3.1).

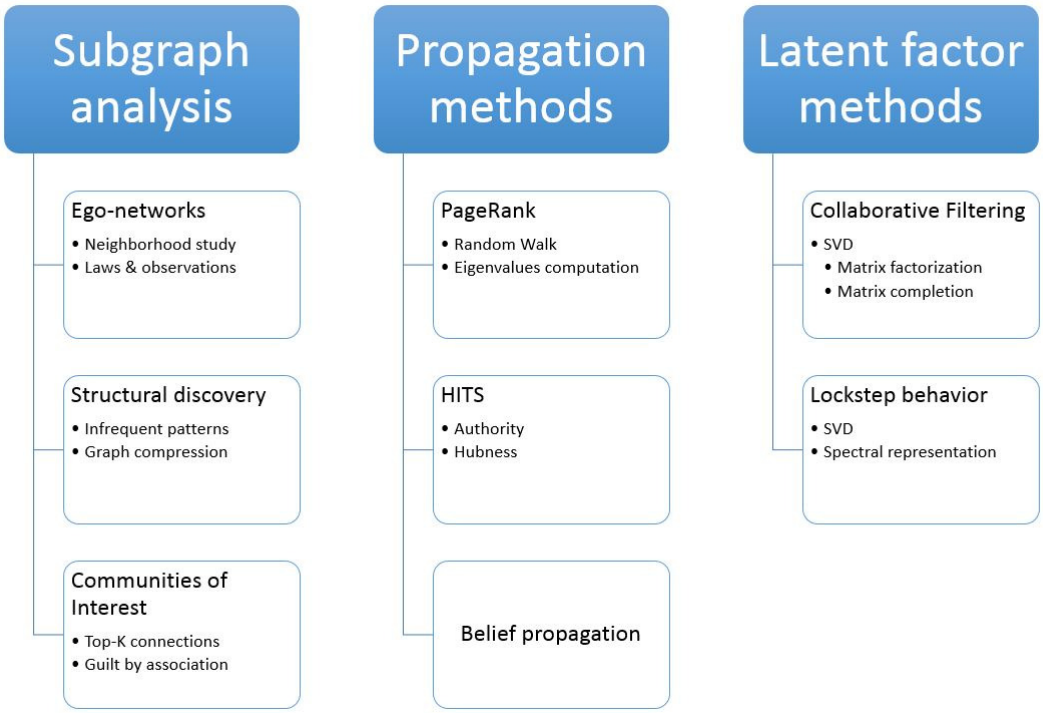

Figure 3.1: User behavior modeling models

3.2 Subgraph analysis

A subgraph is defined as a set of selected nodes in the graph or a specific graph region. We study subgraphs to highlight local properties and detect if they present a divergence compared to the global graph. Significant differences between subgraphs properties and the global graph properties can indicate an abnormal behavior.

One example of commonly used subgraphs is the ego networks (egonets) [4]. An egonet is a subgraph composed of a central node, its neighbors, and all their linked edges.

Akoglu et al. studied in [1] features of egonets to detect anomalies in a graph. They developed a scalable and unsupervised learning method for behavior anomaly detection. The first step of their work computes features that efficiently characterize the nodes' neighborhood. These features are the number of neighbors, the number of edges, the total weight, and the dominant eigenvalue of the weighted adjacency matrix of egonets. The second step combines these features to find normal behavior patterns characterized by these subgraphs. They established a list of laws and observations followed by the majority of the egonets and use outlier detection tools to find anomalous egonets.

Cook et al. used SUBDUE [58] as an anomaly detection method. SUBDUE is a structural discovery algorithm, which finds repetitive patterns in a graph. These patterns are substructures of the graph (e.g., subgraph). Minimum description length (i.e., the lowest number of bits need to encode a piece of data) is used to evaluate the best substructures in order to compress the graph. This is an iterative process, with a phase of selection of the most

repetitive substructure at each iteration.

Let G, S, $DL()$ be respectively the entire graph, a substructure and the description length operation. The following heuristic $F1$ 3.1 is used to choose the best structure at each iteration:

$$F1(S,G) = DL(G|S) + DL(S) \tag{3.1}$$

Let $SIZE()$ and $Instances()$ represent respectively the functions that give the number of vertices and occurrences in the global graph. Starting from the idea behind SUBDUE, they consider anomalies as infrequent substructures in a graph. They use a new heuristic function $F2$ opposite to SUBDUE in order to find unusual patterns (3.2).

$$F2(S,G) = SIZE(S) + Instances(S,G) \tag{3.2}$$

They also proposed another method based on the compression process speed. They started by splitting the original graph into several subgraphs, and they compress each subgraph using the SUBDUE algorithm. Their assumption is that the subgraph which has less compression progression at an advanced step of the process is an anomalous subgraph. This points out that these subgraphs do not possess the most repetitive substructures in the graph. Thus, they are considered suspicious.

Eberle and Holder [25] also proposed a malicious behavior detection process based on SUBDUE. Their system monitors the basic operations on graphs such as the insertions, the modifications, and the nodes and edges' suppressions. Any divergence in the execution scheme of these actions can be considered as proof of suspicious activity. They rebuild the graph starting with a normative structure (i.e., substructure) discovered using SUBDUE. An $insertion$ is the introduction of a new node or an edge. A $suppression$ is the opposite of insertions. It represents the erasure of one node or edges. A $modification$ is the shift of a link or node in the subgraph.

In the telecommunication domain, Cortes et al. [17] introduced principles such as a community of interest and guilt-by-association in order to spot frauds. A community of interest (COI) is a subgraph composed of a central node and the top-K connections (i.e., most used K edges in term of communication exchange) [17]. The guilt-by-association principle is linked to the belief propagation method[73]. We can give the following example of guilt-by-association and belief propagation; If most of the neighborhood nodes of a COI are known as fraudulent, the central node has a high probability of being fraudulent. Cortes et al. assumed that fraudsters have the tendency to be connected and exchange information on their technique. They present a record linkage method based on the belief that identified fraudsters COI can represent a tool to detect new fraudsters. They developed metrics allowing

them to quantify the divergence between two COI. To detect new menaces, they use these metrics to compute the distance between a new COI to the base of the most recent fraudulent communities.

In the financial domain, trading frauds are the main danger. The study of the subgraph topology can be used to detect suspicious patterns. According to [53], spotting unusual topology such as *blackholes* and *volcanos* can be an efficient indicator for trading rings. Blackholes are a group of a node with far more incoming edges then outgoing, which could be indicative of an uncommon purchase of stocks. Volcanos are a group of a node with far more outgoing edges than incoming, which could be indicative of an uncommon sale of stocks.

3.3 Propagation methods

Propagation methods are a family of methods that are useful to spot anomalies on web page distribution (e.g., spam), auction sites, review sites, and social networks. The most known propagation methods are the hyperlink-induced topic search (HITS) [49] and the PageRank (PR) [9] algorithms. The belief propagation [73] principle is also a propagation method. The HITS and PageRank algorithms have the same goal. They determine the most relevant web pages on the internet when there is a query in a search engine. They can be labeled as a ranking algorithm because their main objective is to help web surfers with an ordered list of pages that fit the best to their search topics.

3.3.1 HITS

The HITS algorithm was developed by Jon Kleinberg in 1999 [49]. It uses the links connections between web pages to determine the most relevant pages. According to Kleinberg, HITS is the solution to an abundance problem. A search engine query is a string. The number of pages on the web that could contain the same string is too large for an individual to handle. This highlights the necessity to filter these first search results. HITS uses the authority and the synthesis/hubs relation between pages. The notion of page authority can be defined as the quality of information detained by a web page about the researched topic. Good hub pages can be defined as pages that link to sites with good authority over a searched topic. There is a cyclic relation between the *authoritativeness* and the *hubness*. Pages with good authority scores are pages that are pointed by pages with a good hub score. Pages with good hub scores are pages that are pointing to pages with good authority scores. Hence, for example the *Authoritativeness* 3.3 of a page q is computed by adding the *hubness* 3.4 of all the pages p which are pointing to page q. *Hubness* of a page p is computed using a sum of the *authoritativeness* of the pages q which are proposed by the page p.

Let E be the considered web graph and (p, q) be the node $p \in E$ pointing to node $q \in E$. We have:

$$\textit{Authoritativeness}(q) = \sum_{(p,q)\in E} \textit{Hubness}(p) \tag{3.3}$$

$$\textit{Hubness}(p) = \sum_{(p,q)\in E} \textit{Authoritativeness}(q) \tag{3.4}$$

The page distribution can be represented under a graph format. The HITS algorithm first step consists of creating a small subgraph. This subgraph is built using a textbase search coupled with some page selection criteria to create a small but relevant bucket of pages for the search request [49]. Then, they authority and hub scores are computed.

3.3.2 PageRank

PageRank is an algorithm developed by Larry Page and Sergey Brin, founder of the Google company. It is the first algorithm used by the well-known internet search engine. As the HITS algorithm, it uses information about the links between web pages in order to give a list of the best answers to a search query. The PageRank of a web page can be defined as a score of relevance based on the studies of the transition of a websurfer during a random walk on web pages. A walker (i.e., websurfer) promenades from one page to another page by following the links between two pages. The random walk can be considered as the journey of the websurfer.

As HITS, the distribution of the web pages can be modeled under a graph format and represented under an adjacency matrix. PageRank can be considered as a *Markov chain*, where the pages are states and the transitions are allowed by the links between pages. This leverages the possibility to build transition matrices at each state. The PageRank of a page can be seen as the probability to move from a location on the graph to the designed page at each step of the random walk.

For instance, if we consider the web pages A, B, C, D, and E connected with each other and with unidirectional links in the figure 3.2.

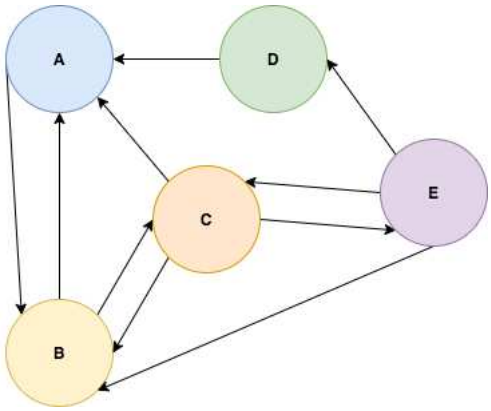

Figure 3.2: Example graph PageRank

Let M (3.3) be the adjacency matrix and P (3.4) be the transition matrix associated with Figure 3.2. We can formalize this example following these definitions.

Let V be a web of 5 pages A, B, C, D, E, X_t be the position of the walker at the time $t = [1, 2, 3, \cdots]$ (i.e., after

$1, 2, 3, \cdots$ clicks) and $Pb(X_{t+1}|X_t)$ be the probability that the walker be on the position of X_{t+1} when he was at time t in the position X_t. The transition matrix 3.4 represents the probability of transition from one position to another after a click. The $Pb(X_{t+1})$ only depend of the $Pb(X_t)$. If initially, the web surfer is on the page A (i.e., $Pb(A) = 1$), after the first click, the probability of the surfer being in the other pages are described in the following.

$$Pb(A) = 0, Pb(B) = 1; Pb(C) = 0; Pb(D) = 0; Pb(E) = 0 \tag{3.5}$$

$$M = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

Figure 3.3: Adjacency matrix

$$P = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{bmatrix} 0 & 1/2 & 1/3 & 1 & 0 \\ 1 & 0 & 1/3 & 0 & 1/3 \\ 0 & 1/2 & 0 & 0 & 1/3 \\ & 0 & 0 & 0 & 1/3 \\ 0 & 0 & 1/3 & 0 & 0 \end{bmatrix} \end{array}$$

Figure 3.4: Transition matrix

If initially, the web surfer is on the page E, at the first step/click we have :

$$Pb(A) = 0; Pb(B) = 1/3; Pb(C) = 1/3; Pb(D) = 1/3; Pb(E) = 0 \tag{3.6}$$

This result corresponds to the columns of the transition matrix. Let the function $L()$ be the number of outbound links. For this example, at any given point in time, the PageRank (PR) of a page A connected to pages B, C, D is given by the following relation:

$$PR(A) = \frac{PR(B)}{L(B)} + \frac{PR(C)}{L(C)} + \frac{PR(D)}{L(D)} \tag{3.7}$$

This is a simplified version of the PageRank algorithm. In practice, it always exists a probability that the surfer falls into a trap of the graph, for example this can be a subgraph very weekly connected to the rest of whole graph. If we consider that the duration of a random walk is infinite, the dominant eigenvalues of the transition matrix highlights the most important pages in the given list (i.e., the dominant eigenvector gives the list of ranked pages). The PageRank can be computed using the power method, which is a well-known eigenvalue problem algorithm.

In order to avoid the trap mentioned below, we consider a general definition of the PageRank approach. Let α be the probability of a user to continue his random walk (also called the damping factor) and $1-\alpha$ the probability of jumping to random pages for avoiding the traps. The PageRank matrix, representing a variant of the random walk model, is defined by(3.8).

Let P corresponds to a $n \times n$ sparse column stochastic transition matrix, G be a dense matrix representing all the graph's nodes giving the equal probability of jumping to each of them with $G(i,j) = \frac{1}{n}$ for all $i,j \in [1,n]$. The improved transition matrix A is defined by:

$$A = \alpha P + (1-\alpha)G \tag{3.8}$$

To determine the PageRank, the vector giving the rank of the nodes, we use the iterative power method to compute its dominant eigenvector x by:

$$Ax = \alpha Px + (1-\alpha)Gx = x \tag{3.9}$$

Statistically, with this process, it is about obtaining the stationary distribution of the graph represented by the matrix A. We can introduce a vector representation of the matrix G by $G = \frac{1}{n}e.e^T$ where $e^T = (1, \cdots, 1)$. Then, an iteration of power method can be expressed as:

$$x_{k+1} = \alpha P x_k + (1-\alpha)\beta e \tag{3.10}$$

where $\beta = \frac{1}{n}\sum_{i=1}^{n} x_k(i)$. The stationary state, allowing to obtain desired eigenvector, will be reached when k tends to infinity.

3.3.3 Hits and PageRank for user behavior modeling

Although these two ranking algorithms have common points (i.e., matrix representation, linear algebra methods), PageRank operates on a more extensive graph and shows better results in a web search. They present applications in the domain of anomaly detection based on the idea that they can detect the most important node in a given graph. As a fraud detection application, a HITS model can be used to determine the node with unusual authority and *hubness* score.

For example, Jiang et al. [46] used graph features to detect fraud in a social network context (i.e., fake account). They consider two types of nodes: source nodes and target nodes. Their method compares these nodes in-degrees and out-degrees respectively to the *Authoritativeness* and the *Hubness* scores. They assess that nodes with good authority scores should logically have high in-degrees. Equivalently nodes with good *hubness* score should have high out-degrees. They establish indicators such as node synchronicity and normality. The synchronicity qualifies

the behavior synchronization of sources and target nodes. The normality evaluates the behavior normality of the target nodes. They also consider the node closeness as an indicator of the similarity between nodes, used to detect abnormal behavior. Behavior anomalies can be detected by visualizing the combination of features and by analyzing the typologies of the resultant models (i.e., out-degree versus *hubness*, in-degree versus *Authoritativeness*).

Gyongyi et al. [38] propose a method to reduce the possibility for a websurfer to go on spam pages. Using the same principle as the PageRank and a seed set of trustworthy nodes chosen by human experts, the authors propose a method that evaluates how trustworthy an internet page is. The *TrustRank* is computed in the same way as the PageRank, but this time only the page closest to the trustworthy set is getting high scores. This reduces the probability of ending up on a spam page. Another feature of the TrustRank algorithm is active when the web surfer stops his random walk and jumps in a non-linked page. The algorithm makes sure the surfer jumps on a page of the set of good pages. This is a particular case of the PageRank methods called *Personal PageRank*. We give more detail about this case in the contribution chapter.

The authors of [20] use the reverse idea of the TrustRank article. They start with a given seed set of spam pages, going backward on the node, and try to identify the group of bad pages at each evaluation of the rank.

3.4 Latent factor models

Latent factor models are the third branch of a method for user behavior modeling. They are mostly based on singular value decomposition (SVD) and the factorization of matrix representing dataset. This factorization highlights the importance of the authoritative scores and creates matrices that numerically model people preferences [4].

The factors and features matrices deduced with the SVD operations give information relative to the original data (e.g., data from a government election period will give information about people's political preferences). One application of the latent factor model is matrix completion. This method, allowing the filling of missing data, is used in the domain of recommendation systems.

The following example is from a work of Beutel et al. on user behavior modeling [4] for the knowledge discovery conference in 2015. It presents the determination of authority and hubness scores showcasing the matrix factorization concept. In this example, the starting adjacency matrix M represents users and the page they like (e.g., users on the rows and pages or items on the columns). The idea behind matrix factorization using SVD is that each eigenvector captures information about affiliation or preference depending on the type of dataset. The relation in adjacency matrix can showcase users and the page they consult, the movies or products they rates, or the people they follow in a social network context.

Starting from known affiliations in a matrix, the SVD based factorization results help to deduce missing values (i.e., matrix completion). With a matrix with missing values, it becomes impossible to find singular values. In this context the use of approximation and a minimization equation (3.11 can be a solution as presented in [3] to complete

Algorithm 1 HITS with SVD & Matrix completion [4]

1: Given an adjacency matrix M build from a graph representation of the dataset.
2: *Authoritativeness* $\vec{v}$ is dominant eigenvector of $M^T M$.
3: $\vec{v} = cM^T M\vec{v}$
4: *Hubness* $\vec{u}$ is dominant eigenvector of MM^T.
5: $\vec{u} = cMM^T\vec{u}$
6: $SVD(M) = U\Sigma V^T$
7: Σ : contains normalization for $\vec{u}$ and $\vec{v}$
8: $UV^T \approx M$
9: $\vec{u_i}.\vec{v_i} \approx M_{i,j}$

the matrix (i.e., estimated values $\widehat{M_{i,j}}$).

$$min(U,V) \sum_{(i,j)\in M} (M_{i,j} - u_i.v_i)^2 \tag{3.11}$$

$$\widehat{M_{i,j}} = \vec{u_i}.\vec{v_i} \tag{3.12}$$

In order to have more precise results, Yehuda Koren in [50] proposed (3.13) to add to the minimization equation latent factors such as μ the datasets means, b_i row user baseline, and b_j column baseline. His goal is to represent the behavior drift of users over time in their preferences. This approach is applied in the context of collaborative filtering, which is one of the main techniques used by recommendation systems.

$$\widehat{M_{i,j}} = \mu + b_i + b_j + \vec{u_i}.\vec{v_i} \tag{3.13}$$

In sale sites and web browsers, collaborative filtering methods are used to produce recommended publicity. An application in fraud detection is the deduction of the possible user activities based on their preference. If user actions follow predicted action using matrix completion, then the behavior will be labeled as usual. If one user behavior is far from his predicted behavior, his actions can be identified as malicious. This idea is close to the hidden Markov models used in [52] and the PageRank.

As another example of a latent factor model, Faloutsos et al. [3] used SVD left and right singular matrix elements (i.e., u_i and v_i respectively in U and V) to detect fraud in social networks. They plot the relation between *followers* and *followees*. Their idea consists to detect lockstep behavior patterns between members of these two groups. Using a spectral visualization, they can spot strange connectivity patterns such and *rays* and *pearls*. These patterns represent uncommon behaviors, hence are anomalies.

3.5 Conclusion

Specifically, for user behavior modeling, work exists in fraud detection research. However, even if similarities can be found between insiders and fraudsters, these techniques can not be directly applied to the organization of company. These graph-based methods are mainly based on anomaly detection or recommendation systems by analyzing graph features. These techniques are different proposition approaches compared to classic anomaly detection tools. However, they share the same issues. Indeed, anomalies can be detected using these methods, but they can't characterize their origins unless. As discussed in the previous chapter, graph-based methods are also used for insider threat in a standalone way [13], or combined with supervised learning methods [33]. However, they are still lots of combination to study, a few of which are proposed in this manuscript.

Chapter 4

Unite & conquer and ensemble learning methods

4.1 Introduction

The tools for user and identity behavior analysis are based on expert-written rules, data science, and some particular methods (e.g., genetic algorithm, decoy documents). The more advanced detection mechanisms are based on the use of data science. However, they usually face the same problem of a high rate of false alarms and non-detection. The root causes of these problems come from the high number of FP/FN of the models suffering from overfitting or underfitting. As discussed before, in order to solve insider threat henceforth the ensemble methods are used in addition to anomaly detection ones. Domain experts propose solutions to deal with these issues. For this purpose, we studied the literature of ensemble learning techniques and the unite and conquer approach.

4.2 Related works to combination of the ensemble learning methods

Machine learning experts mainly combine bagging and boosting to deal with the bias-variance trade-off. Kotsiantis et al. propose [51] a bagging and boosting combination with sum rules voting as the mechanism of collaboration. They build a global ensemble learning method with separated sub-ensembles, respectively, using bagging or boosting. The hypothesis of each sub-ensemble is combined using the voting mechanism to get the final prediction. They tested their solution on 36 well-known datasets from the UCI repository [24] and used decision tree (DT), decision stump (DS), naive Bayes network (NBN), and a rule learner (OneR) as base classifiers. Their proposed technique obtains better results compared to individual bagging and boosting with these base classifiers. Even though this approach solves bias and variance issues individually, this proposition does not manage the trade-off.

In the power plant management domain, [48] propose a parallel combination of bagging and boosting for a regression problem. They used artificial neural networks (ANNs) as base classifiers of their ensemble learning solution for short-term electricity load forecasting. They independently boosted ANNs on different bags of the training dataset. Here the sub-ensembles are all boosted ANNs. The final result of their forecasting is computed by averaging the result of each base method. They compared their effect against ANN, only bagged ANN, and only boosted ANN. They had the best results with their combination of bagging and boosting. In this approach, there is no collaboration between base methods. This could represent a missed opportunity to accelerate the convergence, hence obtaining faster training times.

Fauvel et al. proposed in [31] a hybrid ensemble method called local cascade ensemble (LCE) by combining bagging, boosting, and a mixture of expert (ME) [55] methods in a decision tree scheme. They use their approach for a milk production industry to detect the cow estrus cycle. The ME is a method based on a divide and conquer strategy. This method divides the problem space between classifiers, supervised by a gating network (a weighted average scheme). Each classifier is trained with a different part of the dataset. They use this combination of bagging and boosting to handle the bias-variance trade-off. The diversification properties of ME are leveraged to learn the specificities of parts of the dataset. Their approach showed better results when compared with other classifiers and commercial solutions for estrus detection. Despite the approach interest, it is based on a *divide and conquer strategy*. This strategy is not suitable for the problem that needs global modeling of the training dataset, such as insider threat detection.

These are a couple of examples of methods that combine bagging and boosting techniques. They present strengths such as high bias and variance handling. However, they are missing features to handle the bias and variance trade-off for problems that need an overall characterization of the training data for better accuracy, convergence, and scalability of their implementation. In this manuscript, we propose a new approach to obtain these attributes.

Hall et al. [40] studied the effect of boosting on classifiers trained to detect insider attacks. They create a meta-learner by aggregating boosted classifiers using a probability vote. They tested their method by comparing the base classifiers firstly with their boosted version. They boosted artificial neural network (ANN), naive Bayes network (NBN), support vector classifier (SVC), decision tree (DT), and logistic regression (LR) methods. They obtained mixed results by comparing the accuracy and the AUC of the base classifiers before and after boosting. This process improved the NBN, SVC, DT, and LR slightly, not the ANN and RF, which showed a slight decrease in accuracy and AUC-score. Using their meta-learner, they did not get better efficiency. However, they have a better area under the ROC curve. These mixed results might be a consequence of applying boosting to an already high-performing classifier or not handling the high variance issues by not using a bagging method.

However, it should be noted that in the domain of insider threat detection, there is not any proposition to make a combination of bagging and boosting with a smart collaboration between them.

4.3 Imbalanced data

Data imbalances usually lead to classification methods weak prediction performance. These issues can be identified by using adapted metrics (e.g., precision, recall, f1-score, AUC-score). For machine learning problems in it is very common to be confronted to an imbalanced dataset. Most of the classification algorithms have very bad performances with an unbalanced dataset. In these settings, one class is dominantly present than the others. Hence there is not enough information on the other classes to be correctly classified. This data composition leads to high variance and bias problems. They are two primary strategies to handle the imbalanced dataset, working on the data or working on the algorithm.

Hence, it is possible to handle imbalances by using data sampling techniques such as oversampling and downsampling. Classic random oversampling helps balance the data but adds overfitting risk since the same information is replicated in the data. The K-means clustering method can be used to oversample a dataset by balancing the different data cluster sizes. However, this still might increase the overfitting risk and modify useful frequency information of the samples. Synthetic minority oversampling (SMOTE) is not prone to increase overfitting because the new samples are synthetics and are created from multiple samples. It uses techniques such as the average from two close data-points, KNN, or generative methods. However, the newly created data might not respect the problem specificity. Downsampling might work to re-balance the dataset but at the risk of losing important information.

We can also work on the algorithms using methods that consider the imbalanced nature or modify the data during their training. It exists a family of classification methods called the anomaly detection method. Due to their nature, there are only working with the unbalanced dataset. They consider the dominant class of population or sample as the standard model. The minority classes are considered anomalies. However, these models usage does not specifically handle bias and variance problems and their trade-off.

Variants of ensemble learning methods are proven to be a solution. Boosting and bagging handle, respectively high bias and high variance issues. These methods are capable of modifying the data distribution during the model training phase to get a better prediction performance.

In summary, imbalanced datasets can be addressed using sampling methods or specific techniques such as anomaly detection and ensemble learning techniques. We opt to use the last two to deal with the data imbalances. In this thesis, we propose improving classification results in multiple settings by using ensemble learning techniques. This approach combines bagging and boosting techniques by using an unite and conquer strategy. This approach is based on the collaboration of several boosting methods (i.e., called co-methods) in a bagging context. We describe how a smart combination of such techniques can lead to an efficient solution to improve classification results. In the following sections, we give a presentation of the ensemble learning techniques (section 4.4) and the unite and conquer approach (section 4.5). We propose then a new classification framework based on these techniques in the section 5.4 of this document.

4.4 Ensemble learning methods

The concept of ensemble learning (EL) consists of combining several weak machine learning algorithms. This combination is done to obtain better prediction performances than with an individual learner. For classification problems, a weak learner is considered as a classifier with less than 50% of accuracy. The main idea of ensemble learning is to fuse the information given individually by each learner to enhance the global prediction. These pieces of information are mostly the predicted classes of data samples.

We present here an example from the book of Kumar et al. [74]. Assuming we have an ensemble learning algorithm with *n* classifiers and the error rates of each is $\epsilon = 0.35$. Let us calculate the probability of having m classifiers with the wrong result. Then for $n = 25$ and $m = 13$, the error is approximately equal to 0.06, which is smaller than ϵ.

$$\sum_{i=13}^{25} \binom{25}{i} \epsilon^i (1-\epsilon)^{25-i} \approx 0.06 << \epsilon \tag{4.1}$$

The equation 4.1 shows that it is less likely to have wrong classification results with an ensemble learning method than a single classifier.

There are multiple techniques to apply ensemble learning to improve classification results. The most known methods are bootstrap aggregation (*bagging*) and *boosting*.

4.4.1 Bagging

In bagging, several subsets of the training data are stored in bags using random sampling on the dataset. A given learning method is applied in parallel to these bags. The models are trained with bags of training set and the same testing set. The final result will be chosen in the classification case using hard voting (see figure 4.1). If we take the example of a random forest, randomization is also present in the choice of features to construct the decision trees. The intrinsic parallelism in bagging is a significant advantage. In particular, today, when most of the industrial machine learning problem represents huge data quantities. It is more and more necessary that their processing and analysis be done on parallel and distributed architectures.

4.4.2 Boosting

Boosting is an iterative technique where a set of different weak learners is used sequentially to define a strong learner. Each learner is trained using training data, taking into account the previous learners success. Each sample of the data has an associate sample weight. After each training cycle, the weights are redefined by increasing the ones of the miss-predicted sample. A measure of performance is then calculated for the learner. This process is repeated until the chosen number of iterations (i.e., instances of based method for training) is reached. When all learners are trained and tested, a weighted voting mechanism using the performance metrics is used to establish

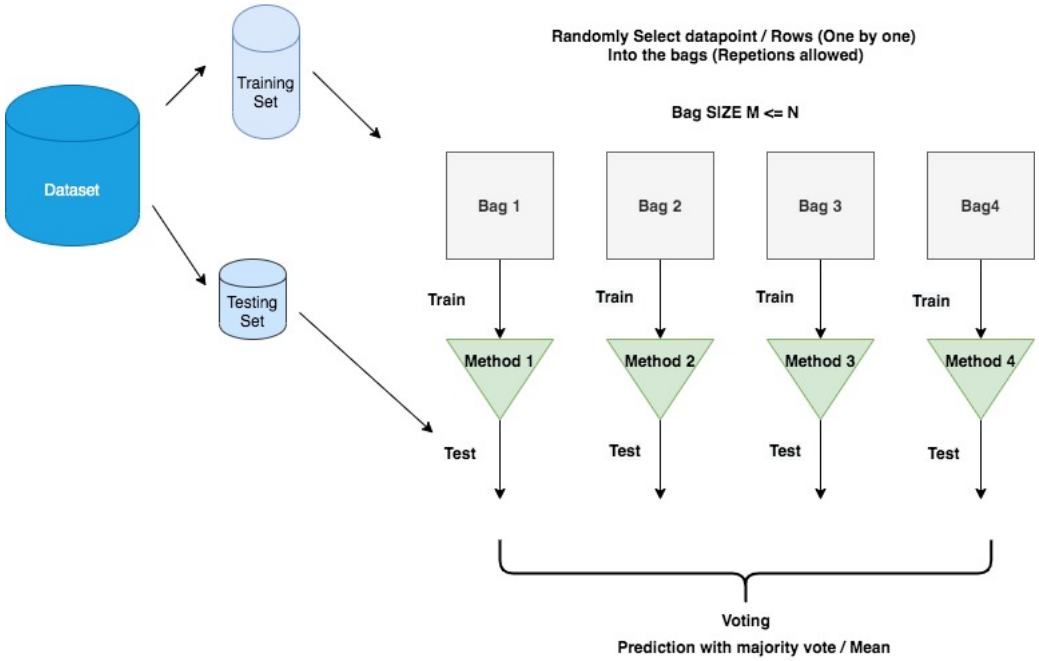

Figure 4.1: Bagging technique

the final prediction (see figure 4.2).

In some case the algorithm stops before the end of the iteration. For example, when the classification error is smaller than a fixed tolerance.

4.4.3 Boosting and bagging algorithms

We suppose that our dataset is decomposed in a learning dataset (LD) and a test dataset (TD) where $\mathbf{card}(TD) = 0$ for unsupervised, $\mathbf{card}(LD) > \mathbf{card}(TD)$ for semi-supervised, and $\mathbf{card}(LD) = \mathbf{card}(TD)$ for supervised learning methods. In the following paragraphs, we use the same terms to designate a learning algorithm and the model resulting from its application to data to simplify the notations.

Let n be the size of LD dataset, ℓ be the number of bags, and L be a set of ℓ learners $L_1, \cdots, L_\ell$[1]. Algorithm 2 represents the bagging technique. Let θ be a threshold from which a model is considered as sufficiently precise,

Algorithm 2 Bagging (in: LD, n, m, ℓ, L; out: prediction)

1: **Start.** Choose ℓ the number of the bags and m ($\leq n$) the size of the bags.
2: **Iterate.** For $i = 1, \cdots, \ell$ do in parallel
3: ***Sampling.*** Select the bag B_i by a random sampling technique with replacement on LD.
4: ***Training and testing.*** Train a model L_i on the bag B_i and test L_i with TD dataset.
5: **Share** On all ℓ processes results, use a selection system (such as voting) to obtain final prediction

[1] Although the machine learning algorithm is different from the learned model (the former uses the data to produce the latter), here we will use the same notation for both.

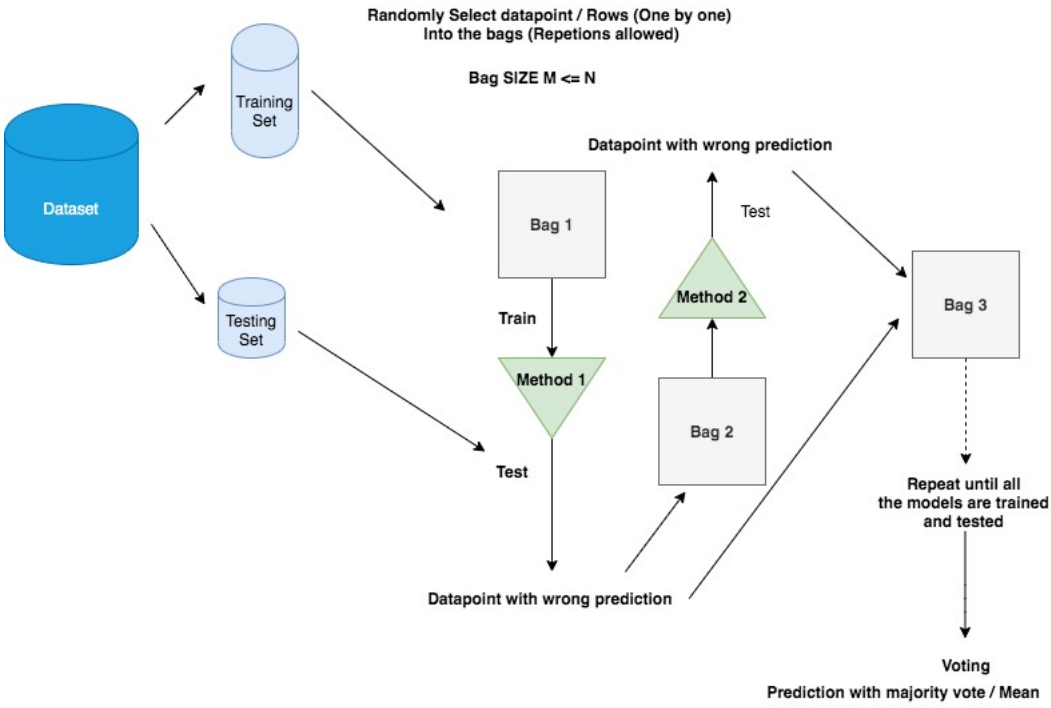

Figure 4.2: Boosting technique

$P(L)$ denotes the precision of the learner L and α be a weight attached to miss-predicted data. We choose α large with respect to the weight of other data in a bag. The following boosting algorithm allows improvement of a training model L_1 throughout the iterations (algorithm 3). Here, all bags are initially chosen as the same and identical to the training data set, but this is configurable and may be different.

Algorithm 3 Boosting (in: LD, n, m, ℓ, L_1, α, θ; out: L_{best}, prediction)

1: **Start.** Choose ℓ, m the the size of the bags ($\leq n$), the base weak leaner L_1 and define the bag $B_1 = LD$.
2: **Iterate.** For $i = 1, \cdots, \ell$ do
3: ***Training and testing.*** Train L_i learner on B_i, produce L_i model, test it on B_i and select W_i the k_i-size miss-predicted sub-dataset of LD.
If ($P(L_i) \geq \theta$) then put $best = i$ and Stop.
4: ***Sampling.*** Define the bag $B_{i+1} = (1 - \alpha_i)R_i \cup \alpha_i W_i$ where α_i the weight given to miss-predicted data and where R_i is the set of $(m - k_i)$ correctly predicted data in B_i and go to 2.
5: **Result.** Define L_{best} as a weighted combination of the l learners.

4.4.4 Bias variance trade-off

Bagging and boosting are specifically used to handle the high bias (i.e., underfitting) and high variance (i.e., overfitting) problems. A classifier suffers from high bias when it is unable to fit the underlying structure of the training set. This results in lousy training and testing accuracy. On the other end, a classifier suffers from high variance

when it is too specialized on the training set. In this case, the training accuracy is high, but testing accuracy is low. Bagging solves the problem of overfitting by slightly diversifying the training set distribution stochastically using bags. Boosting can handle underfitting by specializing a classifier list to a training set. It captures the underlying structure of data by focusing on its most relevant samples to create a model. However, this operation can result in over-specialization on the training set, increasing the overfitting risk. Hence, a combination of the two methods can help to manage the bias-variance trade-off. In this sense, balancing the bias and the variance is the ability for a classifier to generalize beyond its training set correctly.

4.5 Unite and conquer approach

Unite and conquer is an approach initially used in linear algebra to solve large-size sparse linear systems and eigenvalue problems [27]. It permits to build a global solving method made up of several co-methods. This approach consists of making collaborate several iterative methods (also called co-methods) to solve the same problem (see figure 4.3). This process accelerates the convergence of the global method by making use of the intermediate results issued from every iteration of each co-method by all the others. It can be seen as a set of collaborative co-methods that share their restarting cycle parameters to choose the best of them and to reach convergence more quickly. Precisely, this sharing aims to combine the intermediate results to define the best restarting information for each cycle of the co-methods. It allows the global method to reach convergence faster.

Suppose we have to solve a large linear algebra problem P such as an eigenvalue problem or a linear system. Let $L_1, L_2, \ldots, L_\ell$ be a set of iterative methods, each of which can solve the problem P, I_i^k be the initial condition (when $k = 0$) and restarting condition (when $k > 0$) of L_i and, S_i^k be the approximated solution obtained by L_i at the end of its k-th iteration/cycle with I_i^k starting condition (for $i = 1, \cdots, \ell$). The main steps of this approach to solve P are presented in algorithm 4.

Algorithm 4 Unite and Conquer algorithm

1: **Start.** Choose a starting matrix $[I_1^0, \cdots, I_\ell^0]$, let $k = 0$.
2: **Iterate.** For $i = 1, \cdots, \ell$ do in parallel
3: Compute S_i^k by L_i with initial condition I_i^k.
4: If (accuracy of S_i^k is good enough) then STOP.
5: ***Share*** S_i^k information with all other processes j ($j = 1, \cdots, \ell$ and $j \neq i$).
6: **Restart.** Update starting condition $[I_1^{k+1}, \cdots, I_\ell^{k+1}]$ for restarting by $f(S_1^k, \cdots, S_\ell^k)$ and go to 2.

The restarting strategy of UC algorithm is a key component of the global algorithm. It corresponds to the update of the restarting condition I_i^k in function of the obtained results $S_1^k, \cdots, S_\ell^k$ (step 6 of algorithm 4). When the ℓ processes of step 2 of algorithm 4 are running in an asynchronous way, step 5 (*Share*) will be part of "Iterate" (step 2) in the same way as step 6 (*Restart*). In this case, each processes restarting condition is a function of the

most recent available results of all of ℓ processes. The success of the approach depends strongly on the quality of the restarting information at the beginning of each processes new cycle. This information is a combination of the results obtained by all processes in their previous cycle.

In other words, the definition of the function f, which describes this combination in step 6, is of utmost importance for the rapid convergence of the UC algorithm. When the co-methods are the instances of the same iterative method, the corresponding unite and conquer method is also called *multiple X*, where *X* is the name of the co-method.

Recall that an instance of an iterative method represents the method with a given set of parameters (inputs). A brief description of the multiple implicitly/explicitly Krylov restarted methods for solving eigenproblem is presented in [27]. As shown in the papers [28], and [41] in the field of high-performance numerical computing, the asynchronicity of communications between co-methods (processes of algorithm 4), the possibility of overlapping these communications by co-method computations, the fault tolerance, and the potential load balancing are among the attributes of these methods. These properties make them well adapted to parallel and distributed architectures such as GRID, supercomputers, or heterogeneous interconnected networks of supercomputers.

Finally, an essential feature of this approach is its simplicity of the concept. It can be applied to many iterative methods. We will see an example in this thesis with the collaboration between several bagged and boosted methods.

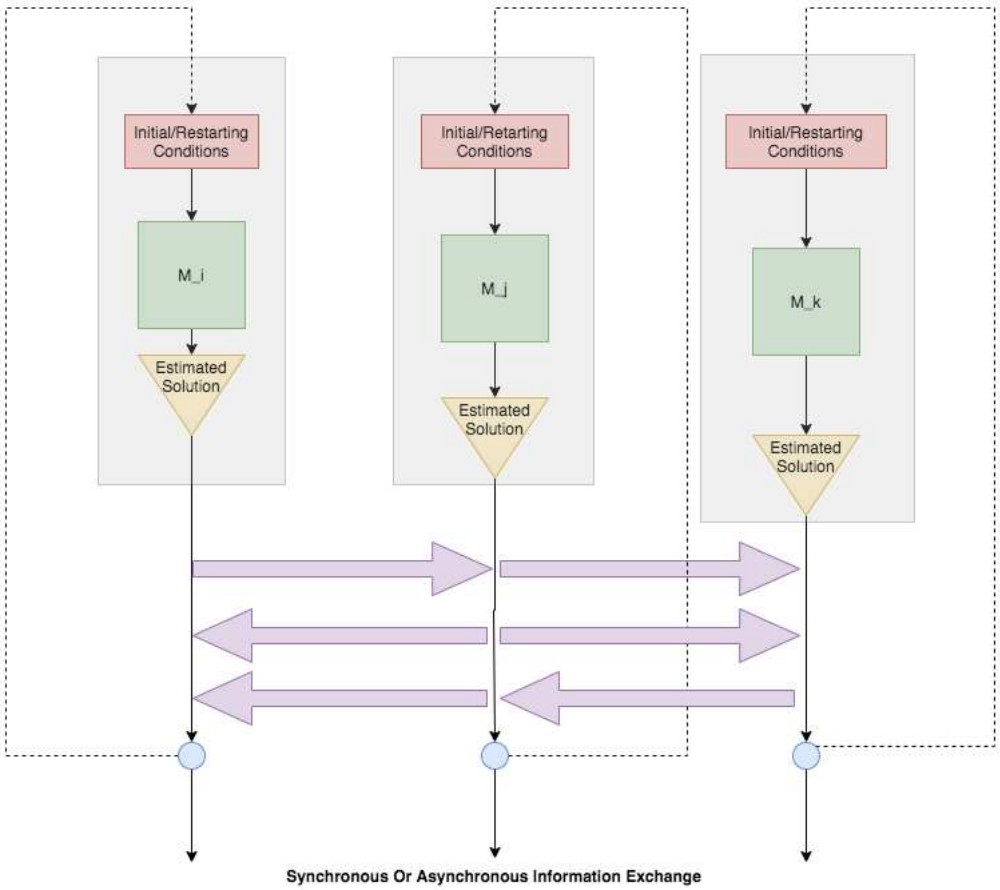

Figure 4.3: Unite and conquer approach for iterative methods

4.6 Conclusion

In this chapter, we explored some basics about ensemble learning and unite and conquer methods. This study allows to highlight the usefulness of the ensemble learning approach with a parallel implementation. We pointed out that classical supervised methods, anomaly detection, and graph-based methods can be used as base classifiers of an ensemble learning method. The bagging and boosting techniques help handle high FP/FN rate issues, overfitting, and underfitting. A framework based on these ensemble methods can also be implemented with high-performance computing techniques. We also presented the unite and conquer method used in linear algebra to accelerate the convergence of the iterative methods. Our contribution uses these fundamental principles to define a new approach dealing with the insider threat detection problem.

Part II

A new insider threat detection framework; from modelling to parallel implementation

Chapter 5

Modelling of the insider threat detection method

5.1 Introduction

In the cybersecurity domain, user and entity behavior analysis software are the tools used to stop insiders. Companies use these tools to determine if employee behavior is normal or abnormal. Employee behavior is hard to classify because it can change over time. Its nature can diverge depending on work requirements and company structures. These tools mainly use behavior classification techniques to detect insiders. Most of the techniques based on data science use supervised machine learning methods and unsupervised anomaly detection. Other approaches use semi-supervised machine learning and graph analysis techniques. Their goal is to detect a divergence from employee usual behavior or to find outliers in the all-around company activity data.

The proposed solutions are diverse, but they usually face the same problem of high FP/FN, and a lack of versatility. Depending on the company studied, a method initially performing well to detect an attack scenario can present an unstable detection accuracy. It makes sense to optimize a specific model for a particular company, but the cost of the software development and maintenance of these tools can represent a drawback. The data volume can also pose a challenge to implement a detection model. Machines limited in their computation power struggle to treat the massive amount of behavior data. A solution to this issue would be to build a detection model somehow adaptive and able to manage different and extensive data input. This model would have to consider detection accuracy, detection time, maintainability, and versatility constraints.

This chapter focuses on the presentation of the proposed framework model, the corresponding algorithms and its comparison with similar methods.

5.2 Framework proposition basis

Insider threat detection and fraud detection using behavior modeling can be seen as a data science binary classification or as an anomaly detection problem. In an activity dataset, there are mainly normal or abnormal behavior events. For many data science problems, classification methods are often the solution. They belong to the family of pattern recognition methods and are mostly used for supervised learning problems.

Statistically, classification methods identify the members of sub-populations in the data. Classification methods performance bottleneck can be a statistical issue such as a class imbalance in the training data, poor features selection, and imprecise hyper-parameter selection. If we focus on class imbalances, the use of anomaly detection methods might be a solution. These methods are built to treat imbalanced datasets, but they might suffer from bias and variance issues.

We mostly use ensemble learning methods such as bagging and boosting to fix these statistical issues. However, individually used, these methods do not consider the bias-variance trade-off. We discussed in the previous chapter that fixing high bias issues with boosting can lead to a high variance problem.

Thus, to propose a better solution to remedy these issues, we developed a parallel and scalable machine learning framework. The proposed approach is based on the use of unite and conquer and the ensemble learning principles. This solution uses the collaboration of multiple machine learning methods to build individual post-login activity profiles. These profiles are used to identify a new activity record as regular or abnormal.

This framework combines base learners iteratively using boosting and bagging according to a unite and conquer approach from the linear algebra domain. This combination helps manage the bias-variance trade-off and accelerates the convergence of the ensemble method training phase. Statistically, this is an iterative approach that modifies the training data multivariate distribution at the beginning of each cycle to improve previous classification results.

The proposed technique also has the advantages of using multiple types of classifiers, and its implementation is scalable. The base learners are supervised ones, anomaly detection methods, unsupervised methods, and graph analysis techniques. This versatility helps to deal with multiple data types and compositions. The framework produces an optimized classifier or a new ensemble classifier. We named this framework UCEL (for unite and conquer applied to ensemble learning).

We show that global method makes it possible to obtain significant accuracy with respect to the base methods (co-methods), which constitute the global method proposed. To exploit the proposed solution potential parallelism, we implement it with high-performance parallel computing techniques and show its efficiency. In some of its versions, using co-methods such as graph-based analysis with PageRank and the recurrent neural network with an autoencoder, UCEL offers new techniques for insider threat detection.

These two methods are the result of our study of graph-based fraud detection techniques, time series, and recurrent neural networks. We used them as base learners of the UCEL framework.

We also propose a container-based insider threat detection modules model. This choice of architecture ensures our detection systems portability to different organizations on-premises structure and cloud systems. This module is enriched by alarm systems that scan daily user activities. This system uses all the trained behavior profiles to establish an activity normality score based on the classification of new unlabeled events. The modeling aspect of the framework proposed here has been published in [21, 23], and some of its high-performance computing aspects in [22].

The present thesis offers a complete analysis of this new framework starting with the fundamental elements necessary for its modeling and going up to the experimental results allowing its validation. It also goes through the preprocessing of the data, the proposal of suitable parallel programming models, the new learners based on graph analysis and RNN, and the insider threat detection.

5.3 Detailed contribution

In this work, we propose an insider threat detection solution based on the ensemble learning paradigm and a customized combination of bagging and boosting. This contribution is inspired by the previous work in the insider threat detection, fraud detection, behavior modeling domains, and the mitigation of high bias and high variance problems. For the bagging and boosting combination, we opt for the same strategy as the ME method. However, we diversify the distribution of the training dataset using bagging and boosting sampling techniques to handle the bias-variance trade-off. So contrary to a divide and conquer strategy like ME, we propose a restarting strategy based on the unite and conquer approach, mixing individual classifier classification feedback to improve the training set. Here we summarize the contributions of the present thesis:

- Proposition of a new and efficient iterative machine learning framework, based on the application of unite and conquer approach on ensemble learning technique. This method can be seen as a particular combination of boosting and bagging with a restarting strategy.
  - From the methodological point of view, this method offers means to accelerate the convergence of many classical ensemble learning algorithms and an algorithmic solution to remedy imbalanced dataset issues in applications such as insider threat detection.
  - From the point of view of calculation, the proposed method offers intrinsic heterogeneity, potential load balancing, fault tolerance, and the possibility of overlapping communications with computations, which makes it very well suited to today emerging high-performance architectures.
- A high-performance insider threat detection software framework, providing user and role-based behavior profiling based on supervised, anomaly detection, graph-based, and RNN machine learners;

- A component library combining anomaly detection and supervised learning methods depending on the available data;
- A component library for an insider threat detection method using graph analysis and the PageRank Algorithm;
- A component library for insider threat detection techniques based on the use of an auto-encoder with RNN LSTM cells;
- A portable model and its implementation of an insider threat detection module with corresponding alarm systems based on docker containers.

5.4 Proposed framework overview: UCEL

Our first goal was to build a global learner composed of classifiers as co-methods using the UC approach and the EL bagging and boosting techniques to improve performance. To this end, we propose to apply the unite and conquer approach to ensemble learning techniques. In the UC context, based learners can be seen as co-methods and bags as part of their initial parameters.

Unlike ensemble learning methods, we use a restarting strategy based on the oversampling of FP/FN. This oversampled imformation is issued from the current iteration and can be used to improve the model for the next iteration.

The robustness of the global leaner is due to the relevance of this restarting strategy. We call the framework defining this global learner UCEL (for unite and conquer ensemble Learning) (see figure 5.1). UCEL is composed of several machine learning methods that collaborate in order to acquire information from data. Its goal is to produce precise models more quickly than with individual methods.

The combination of methods in machine learning presents an important rank of possibilities, as we can notice in [21, 81, 7]. The combination of bagging and boosting using a unite and conquer strategy belongs to the family of learning methods that can manage bias and variance trade-off. This aggregate can be seen as the parallel execution of several boosted co-methods, with training data created with bagging methods. Additionally, boosted co-methods exchange their oversampled intermediate dataset to improve the restarting conditions of each cycle of the co-methods.

5.5 UCEL mechanisms

A co-method starts with the initial conditions arbitrarily chosen before the first iteration. The initial condition for classifiers are hyperparameters and the training set. At the end of each iteration/cycle, the conjunction of the

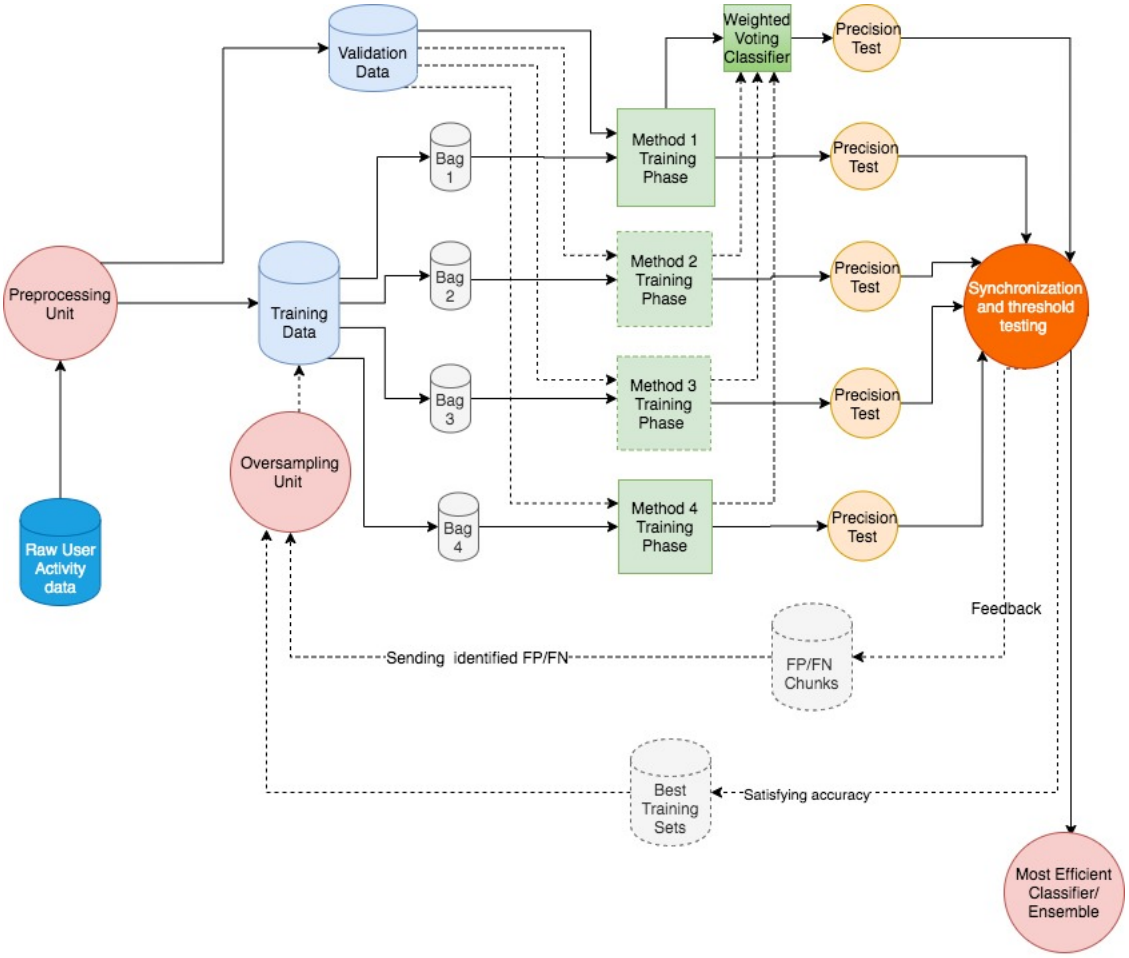

Figure 5.1: UCEL framework model

intermediate results is used to define new starting conditions for the next cycle of co-methods.

The intermediate results are essentially the training bag and the wrongly classified elements. In other words, after an iteration, the acquired experience of each classifier is used to improve the restarting conditions and, consequently, the classifier. It is the same as the mechanism of boosting. However, it is applied here with collaboration between the co-methods.

Each co-method is trained with a bag of the training set. Their performances are tested with the validation set. Following some boosting principles, each time a sample from the validation set is misclassified by a co-method, its weight is increased based on the co-method performance metric at that iteration. According to this weight, the sample chances to be selected to build the next cycle training data are increased. Hence, a co-method weighs and shares its misclassified data FP/FN sample to the rest of the co-methods to create new training bags. These bags are composed of the former data and the oversampled misclassified data with the highest weights. With this process, each co-method receives feedback on the data samples critical to classify by their peers correctly, to build better models.

The injection of the wrongly classified elements to the training set modifies its distribution slightly. This mod-

ification impacts the prediction of the model. This approach is common to the semi-supervised technique, which improves the training data distribution by introducing labeled data. However, in the UCEL context, this is done iteratively, with several learners and using a strategy to accelerate the convergence. The wrongly classified data samples are pieces of information that are used to build new, improved bags of training data. Besides, these data are oversampled using random or synthetic minority oversampling. This operation gives more data to the learners to build efficient models but still considers bias-variance constraints with sample weights. Hence, this approach allows the global method to reach convergence faster than a classic boosting iterative process.

The exchange of information can be carried out synchronously or asynchronously. In the latter case in the context of parallel computing, the calculations and communications can overlap and cause significant time savings. Unlike the classic bagging method in ML, the bagging operation does not stop after a single cycle but after obtaining the target accuracy for co-methods or the fixed number of iterations. It should be noted that when the same method is used several times in UCEL, the co-methods are the instances of the same learner. This represents multiple UCEL methods (MUCEL), which is a particular case of UCEL.

5.6 UCEL algorithm

Let $M = [m_1, \cdots, m_\ell]$ be a set of ℓ integers ($m_i \leq n$ for $i \in [1, \ell]$) representing size of the bags, $L^0 = [L_1^0, \cdots, L_\ell^0]$ be the initial weak learners, $B^1 = [B_1^1, \cdots, B_\ell^1]$ be a set of ℓ initial bags and q be an integer representing the number of iteration of the framework (i.e., maximum number of boosting phase). An algorithm of this approach for the case of classification is given by algorithm 5.6 where LD and VD denote learning and validation datasets respectively.

V is a function that creates a weighted voting classifier and f a function that selects $(B_{best}^j, L_{best}^j, W_{best}^j)$ as the best results of each co-method received from all $i \in [1, \ell]$ processes.

Obviously the best information of interest denoted as $(B_{best}^j, L_{best}^j, W_{best}^j)$ in the step 5 of the algorithm 5.6 depends on the function f defining the "best" of (B_i^j, L_i^j, W_i^j) for $i \in [1, \ell]$. It should be noted that when the UCEL algorithm is in asynchronous mode, the iterations of the boosted co-methods are not synchronized. Consequently, at the end of an iteration of a boosted co-method, the function f has to select the best intermediate results available at that moment. It is to note that when the ℓ UCEL's processes run in synchronous mode, the number q_i of boosted-learners has to be the same for all $i \in [1, \ell]$.

This algorithm has essential features that makes it attractive in the accuracy of the prediction and parallel programming efficiency. Indeed, the possible heterogeneity of the co-methods composing it implies a load balancing potential. The possibility of asynchronous communications between co-methods allows overlapping of these communications with calculations. The UCEL algorithm is fault-tolerant; the disappearance of one or more co-methods does not prevent its functioning as long as one of the co-methods is working. UCEL presents multilevel parallelism, including a coarse grain inter co-method parallelism and another finer grain intra co-methods. However, it

Algorithm 5 UCEL (in: LD, VD, n, ℓ, M, B^1, L^0, θ; out: B_{best}, L_{best}))

1: **Start.** Choose ℓ the number of the bags, M the size of the bags, L^0 the ℓ learners and

2: **Iterate.** For $i = 1, \cdots, \ell$ do
3: **Iterate.** For $j = 1, \cdots, q$ do
4: ***Training and testing***
Train L_i^{j-1} on B_i^j, produce L_i^j, test L_i^j on VD and select W_i^j.
5: ***Sharing informations***
share (B_i^j, L_i^j, W_i^j, AUC-score(L_i^j)) with all ℓ co-methods
6: ***Compute WVC and stopping test*** $WVC_j = V(L_i^j, AUC(L_i^j))$
$B_{best}^j, L_{best}^j, W_{best}^j = f(L_i^j, B_i^j, W_i^j, WVC_j)$
If (AUC-score(L_{best}^j) $> \theta$) then STOP all processes.
7: ***Oversampling***
Set the bag $B_i^{j+1} = (1-\alpha) * OvS(W_{best}^j) \cup (\alpha) * R_i^j$ where R_i^j is the set of $(m_i - k_i^j)$ correctly predicted data in B_{best}^j with $k_i^j = \mathbf{card}(W_{best}^j)$, α is the updated weight given to miss-predicted data,and OvS the oversampling function.
8: ***Result.***
Set L_{best} the best individual co-method or best weighted combination of co-methods during the iterations of all ℓ processes.

should be noted that parallel programming of this algorithm is a delicate task and its implementation, particularly in asynchronous mode, needs great attention.

The function f of the step 5 defines the restating strategy, which is a key element of the algorithm 4, uses the information of interest generated by co-methods to establish the global new restarting conditions. Obviously, there are many possibilities to define this function. In this work, the function f is defined as the following:

$$(B_{best}, L_{best}, W_{best}) = \underset{AUC_score(L_i)}{\arg\max} ((B_1, L_1, W_1), \cdots, (B_\ell, L_\ell, W_\ell)) \tag{5.1}$$

The weighted voting classifier (WVC) is an ensemble learner issued from a weighted voting process to classify data. It combines all the base learners $L_1, \cdots, L_\ell$ and uses their corresponding AUC_score as their weights. The popularity function identifies the most common misclassified elements for the next training cycles. We use sampling techniques at this level.

In summary, the UCEL algorithm is a global semi-supervised training method that improves each co-method classification performance during its iterations, using an information exchange mechanism. This process is repeated until one of the co-methods reaches a convergence state. This state will be reached with enhancement and stabilization of individual performances or by reaching a fixed number of iterations.

5.6.1 UCEL comparison with a mixture of experts

We can compare the UCEL approach to a classic divide and conquer based methods such as a mixture of experts. ME is an application of a divide and conquer strategy followed by a weighted voting scheme (a probabilistic gating function [55]) to classify new data. This means that the problem space is divided into homogeneous parts, and there is no iterative improvement of a base method. Unite and conquer learns the underlying global structure of data, where divide and conquer approaches learn the underlying substructure since the data is strictly divided. Depending on the machine learning problem, these different characteristics might be an advantage. For example, in an insider threat detection problem, we want to use insights from each co-method on a dataset that have the same statistical properties to fit a global behavior. In this case, it is not suitable to strictly divide the data [23].

We emphasize the fact that the UCEL approach can be considered as an improvement of ME. UCEL switches the classic expert's consensus methods to predict a sample class by introducing collaboration between methods. This collaboration mechanism is based on the statistical data diversification of bagging and boosting, sampling techniques, multiple training cycles, and the convergence acceleration of each co-methods. The combination of bagging and boosting manages the bias and variance trade-off since the training bags are built using a fusion of all the co-methods feedback information.

ME results would be almost equivalent to the first cycle results of UCEL approach if we did not apply bagging and strictly divided the dataset. Lastly, it is essential to note that the error is individual to each base method with a divide and conquer approach. With UCEL, each classifier is somehow responsible for the global error. This approach has a global loss function that we optimize by improving the training data distributions. Hence to decrease the global error, every co-method needs to be tweaked.

5.7 UCEL co-methods

Depending on the nature of the problem, a UCEL framework can be built using supervised, anomaly detection, or graph-based method as co-methods. Bagging and boosting are mostly used to deal with a supervised learning problem. However, the ensemble learning concept can be applied to different categories of machine learning methods. Since UCEL is an improvement of an ensemble learning method, it inherits his capacity to work with multiple types of methods. The nature of the machine learning problem and the properties of the studied dataset can motivate the kind of base methods used within UCEL.

In a case where the dataset of our problem has a natural class imbalance, which means that there are many more negative samples than positive ones (or the other way around), anomaly detection is the go-to solution. Unsupervised anomaly detection methods are tuned to work with imbalanced data. They are the most suited to deal with these issues due to their working mechanism. The majority class is set as the main behavior model of the

data. Everything distant from the normal activity sample distribution, regarding the mean and the standard deviation, is considered abnormal.

We can consider the anomaly detection problem by considering a dataset of normal samples as a *Gaussian* distribution of $x_1, x_2, x_3, ...x_n$ samples. With this distribution, we calculate $p(x_i)$ as the probability that a new sample data x_i belongs to the normal distribution of data. For example, if $p(x_i) < \epsilon$ with ϵ a given threshold, then x_i can be considered as an abnormal activity.

For example, let us consider an insider threat detection problem again. Negative samples are employee usual activities, and positive samples can be seen as a change in his/her everyday activities or work practices. Precisely for this problem, if a dataset of activities is constructed, it is usually constituted by far more normal actions than abnormal actions. Indeed, due to the lack of positive samples, supervised binary classification methods might be inefficient for this kind of dataset. This is because they are not disposing of enough information to learn the characteristics of positive examples.

Hence, it is hard to establish a decision boundary to distinguish classes. We discussed before that a remedy to imbalanced datasets might be to balance them using over/undersampling techniques. However, specifically for this security example, new samples may not provide additional information or can be too synthetic to represent an actual attack scenario accurately. Moreover, in practice, labeled data may not be available and may be challenging to obtain. New data samples are generally not labeled.

In the overall literature, supervised learning approaches showcase better performance than anomaly detection approaches. Particularly approaches based on neural networks. However, to work properly, they need substantial and balanced data. Hence this method can struggle to perform well and can be susceptible to bias and variance issues.

Graph-based methods are another approach to tackle machine learning problems. Based on the graph features analysis, the detection mechanism is to spot anomalous nodes or subgraphs by computing their features (node degree, PageRank, link distance). This approach shares the same issue as anomaly detection approaches. They are also sensitive to exceptions, and they do not give information on the nature of the anomaly.

Using UCEL with these co-methods helps to use their diverse capabilities and advantages. This diversification helps to tackle a problem from multiple angles. It also helps to maximize the chance to find the most appropriate solution. Additionally, this approach can also serve as a showcase of different resolution methods of the same problem.

In conclusion, regardless of the family of co-method, after performing data preprocessing, we can apply the principle of UCEL. However, the type of exchanged information can be different. Additionally, to the wrongly classified samples and the original training data, the best hyperparameters chosen by UCEL can also accelerate the convergence.

5.8 UCEL for insider threat detection

In order to solve the problem of insider attacks, machine learning methods are used to analyze employee activity data. This data is used to model employees behavior by learning from their post-login data (i.e., after logging in to a laptop or company server). Depending on the type of data available (labeled, unlabeled, or graph-based), we can use a variety of suitable methods. They can be supervised classification methods, unsupervised classification methods (based on distance, density, or hierarchical grouping), anomaly detection methods, or methods based on graph analysis. These methods help to create employee-specific, role, or company behavioral profilers and activity analyzers.

In a corporate environment, the vast majority of employees are considered not insiders. There is a natural class imbalance in their activity data. This means that there are many more samples of good activities recorded than there are of malicious activities. For example, if we focus on a single employee, insider action can be seen as a change in his/her usual work practices. If we register for an extended period the post-login activities of companies employees, we will most likely have an imbalanced dataset.

With this noticeable class imbalance, anomaly detection methods seem to be the most suitable for identifying insider activity. Indeed, due to the lack of abnormal examples, purely unsupervised and supervised classification methods might be inefficient for this kind of dataset. They are not disposing of enough information to learn the characteristics of abnormal examples. Hence, it is hard to establish a decision boundary to distinguish between normal and abnormal activities.

However, for this problem, anomaly detection approaches usually suffer from high FP/FN rates and only focus on detecting what is unusual in the dataset. Consequently, they do not precisely characterize the cause of the attack. Human action is needed to determine the causes of abnormalities. On the other end, supervised methods can be trained to detect anomalies related to the specific types of insider threat attacks, but as we stated before, they need a balanced and labeled dataset to perform well. Moreover, the same supervised classifier is not necessarily efficient in spotting different types of insider threat attacks. Hence, human action is required to label the data as standard and to choose and tune supervised classifiers.

Since the flow of activity data is continuous, labeling activity and balancing data before testing for insider threat might be more risky and costly for companies than characterizing anomalies when found. Hence, in the context of UCEL, we propose to opt at first for behavior profilers using unsupervised and semi-supervised anomaly detection. We can then combine them with a supervised learning profiler when we have enough labeled and balanced data. This is by using human operators to label the data or oversampling strategies [70] to deal with the imbalances.

In a semi-supervised context, [36], anomaly detection methods add samples with known labels to their data distribution to have extra information to their classification process. This action improves the decision boundary of classic unsupervised anomaly detection methods [39]. A behavior profiler with anomaly detection as co-methods

can be alimented with a continuous feed of FP/FN samples, labeled by another security system or human action.

5.8.1 UCEL from data to behavior profiling

Before analyzing employee behavior by the proposed profilers, it is necessary to perform a data preprocessing step. In this step, we start by selecting samples from a single employee raw activity data and samples from insider threat attack scenarios databases. We then perform classic feature engineering with dataset cleaning, feature selection, scaling, and normalization. We finish by building a training, validation, and testing set from the cleaned data. We give more details about the training split in the experiment part of this document III.

The second step is to design our behavior profiler using UCEL. In the anomaly detection case, the co-methods or base methods can be mainstream methods such as IForest, OcSVM, robust covariance (Robcov), and local outliers factor (LOF). We can also choose co-methods in the supervised learning case, such as multilayer perceptron (MLP), Gaussian naive Bayes (GNB), KNN, SVC, and graph-based methods. We can also build a solution using the same method but with different hyperparameters (i.e., starting conditions). In other words, we can use the instances of the same base method, which is equivalent to create the particular case of the UCEL methods called multiple base methods such as multiple IForest or multiple Robcov. However, the advantage of using co-methods differently is that it helps build a heterogeneous consensus on the nature hypothesis of an analyzed behavior.

It is important to note that none of the above-cited methods individually can be considered as well adapted to all the situations. Each has unique advantages and disadvantages and can match the specificities of the companies activities. Their combination using the UCEL framework allows enjoying the benefits of these methods without their drawbacks. Hence, the co-methods are chosen to be classic anomaly detection methods or supervised classification.

In UCEL context, we start the first iteration by training the co-methods with bags generated with an RSR on the initial training data. We then test the co-methods with the validation set by computing the AUC-score of each co-methods. After that step, we then try to combine the strength of each co-method and build a weighted voting classifier (WVC) with their results. We then test its AUC-score against the score of the co-methods and the chosen detection threshold.

If the detection threshold is not reached, a new cycle begins with the application of boosting on the previous training bag. For that, we start by gathering all the FP/FN of the co-methods and weigh them in the function of their popularity. The training data of a new cycle of a co-method is obtained by combining the most accurate training bag during the previous iteration with the most common FP/FN oversampled. Here, we use a WRS and a RSR again to build the new training bags. All the best bags and WVCs are stored for their future using in the following iterations. They are updated when a new bag presents a better prediction score.

This process is repeated until the detection threshold or the desired number of iteration is reached. The restarting

strategy, which is a critical part of the proposed model, defines the conditions of the running of a new cycle of classifier training and testing. In theory, the new cycle has better starting conditions than the previous ones. Here we make use of the above-cited simple restarting strategy. Still, the proposition of the ones with more effectiveness and sophistication will be the subject of our future research works. In addition to the best bag, we can consider sharing the best hyperparameters in a multiple-UCEL case.

Depending on input data, even with the boosting process, some co-methods might not be efficient to detect the insiders. Hence, they may not contributing to accelerate the convergence through the iteration. In those cases, UCEL still provides good results because the focus is always shifted to the best method and the best bags. In some sense, this highlights the fault tolerance characteristic of UCEL, which is defined by its proper functioning despite the possible disappearance of certain co-methods, as long as one of them remains. The bags are stochastically selected. Hence, in the same conditions, the performance might also drop if the selection is unlucky at that cycle. However, since we choose the best bag since the first iteration, combined with the situation where the classifier is mistaken, the precision is susceptible to oscillate and rise again. The stopping condition for unite and conquer methods consists of reaching a chosen tolerance. The proposed stopping condition of the UCEL system here is to have a base classifier AUC-score reaching a chosen threshold. At that point, the best-trained co-method or the best-weighted voting classifier is considered to be *the behavior profile*.

5.8.2 User profile vs. Role profile

There are some notable divergences between the user and role profile. User profiles are built exclusively using subsets from the targeted employee activities data. They allow comparing users new actions to their past activities. Role profiles are built using a larger subset of the targeted employees and their peers (i.e., colleagues with the same roles) activities. They allow comparing new actions to their peer actions.

If we focus on identity access management and insider threat problems, it looks more advantageous to build a more general detection solution. For example, if we create behavior profiler for each employee, the number of the profile created can be significant depending on companies size. Since insider activities are usually out of the scope of role prerogatives, it is practical to classify users by their roles and check if they are doing something out of the scope of the role. However, using a role-based behavior profile can result from drowning the specificities of the user. For example, if the attack is linked to the volume of activities, a role dataset might trouble the detection method since it focuses on average role activity volume.

Since there are more users than roles, building an important amount of user profiles needs more computation resources than building role profiles. It might be redundant to build a profile with a user having the same role. Hence, role-based insider threat detection seems to be the most practical solution. However, it might miss some insider attacks by not focusing on user-specific activities.

In conclusion, the best strategy is to use both, to have two approaches of detection. An insider is detected if there is a significant difference in a user behavior from his profile or the activity patterns of his colleagues. The behavior profilers are the UCEL framework instance or the independent classification or anomaly detection methods. In the case there is a divergence in the employee activity volumes, or frequency, a user profile will detect the attack. In an extreme case where the behavior profile was built from data where the individual behaved systematically in a malicious way, a comparison with his peers should mark him as suspicious. It is also the case where he has just been recruited. This approach might use more hardware resources, but it provides a more global protection.

5.9 Conclusion

In this chapter, we presented a synthesis of our contribution in terms of modeling of insider threat detection systems. We essentially discussed the establishment of user behavior profiler using the UCEL framework. Its specificities will be disscussed in details in the validation part III including the presentation of the PageRank and autoencoder techniques as co-methods for UCEL framework.

Chapter 6

Integration of serial and parallel UCEL in an insider threat detection system

6.1 Introduction

We present some specificities of parallel implementation of the UCEL framework and the insider threat detection modules. We start by giving a synthetic presentation of the main parallel programming models for the UCEL framework, according to a component approach, and allowing maximum exploitation of its potential parallelism. We implemented UCEL according to these programming models integrating components such as libraries, datasets, etc. We also present the integration of UCEL to a global insider detection system.

6.2 UCEL parallel programming paradigms

The parallel implementation of ensemble learning methods, by its bagging component, is straightforward, but we can go further using the UCEL framework. By considering the UCEL algorithm 5.6, we can see immediately that the outer loop (steps 2) can run in a parallel loop that performs coarse-grained calculations. Steps 5 is the step of sharing the feedback of the co-methods. A parallel model that assigns the computation unit (node, core) to the co-method could use a communication mechanism (synchronous or asynchronous) to share the intermediate results.

A critical aspect of this algorithm is its possibility of asynchronous communications in step 5. This means that after executing an iteration, a co-method does not necessarily have to wait for synchronization with other co-methods before starting the next iteration. This allows overlapping of these inter co-method communications by the computation of base-classifiers and a significant saving of overall execution time. That being said, these communications

inter co-methods can also be done in synchronous mode. Despite a loss of time due to this synchronization, the advantage of this mode of communication is the simplicity of its implementation and the contribution of a certain determinism in the calculations.

Another important aspect of the algorithm is its heterogeneity. Indeed, the processes of steps 2 corresponding to base-classifiers could all be different. Thereby, it is possible to use an adapted hardware processor for each of them according to their natural parallelism.

For instance, it would be more optimized to use a multi-core node for an *IForest* method and a GPU architecture for a neural network whose algorithm can present a high degree of data parallelism. Thereby, at least two-level parallelism is present in this algorithm; inter and intra co-methods. Besides, the natural parallelism of the co-methods allows load balancing of the system by assigning each of them to suitable hardware architecture. That is also to note that the algorithm is fault tolerant. Indeed, the disappearance of a co-method by any fault does not prevent the other co-methods from working and making the algorithm work up to its end.

One of the essential criteria to implement an efficient parallel algorithm is the optimization of communication between its components. Indeed, generally, the execution time is dominated by the time of communication. We assume that the targeted parallel architecture has a set of nodes. Each node could be itself a parallel processor such as a multi-core processor, a vector processor, a GPU processor, or even a serial machine. For the parallel implementation of this algorithm, two main parallel programming models based on optimization of communication can be targeted.

Detailed analysis, as well as many experiment results of parallelism of UC methods which represent the same context as UCEL framework, are given in [27, 29, 15, 41, 28].

6.3 Client-server parallel programming model

Thereby, each co-methods can be given to the available core, and this up to the achievement of convergence. Available core computes an iteration of the co-method, they communicate their results to the server node. At the end of a cycle, all co-methods receiving information of interest from other co-methods (FP/FN). They use this information and updates their initial conditions for restarting another cycle.

For the first model we propose is a client-server paradigm 6 where the iterations are share with the available computation resources.

In this case, the processes cannot communicate directly with each other. With this model-based inspired by a client-server one, the nodes act as a server, a client, or an agent/controller. A server provides resources and services to one or more clients through one agent or controller. We consider a system with a set of servers, one client, and one agent represented by nodes of our targeted parallel architecture. It is to highlight that this parallel programming model is particularly well adapted to distributed systems where client, server, or agent can be

represented by any independent computer integrated into the system. An advantage of this programming model on such a system is its capacity to provide access to otherwise unavailable resources by making the power of supercomputers accessible from low-end machines like laptop computers as servers.

In the case of the proposed parallel behavior profiler algorithm, for each cycle of a classifier, the actions to be performed are (1) client contacts the agent for a list of capable servers, (2) client contacts computing server and sends input parameters, (3) server runs appropriate service and, (4) server returns output parameters or error status to the client through an agent. A server can not keep control of a co-method or classifier until the convergence. Indeed, a server calculates an iteration of a co-method, but it is not at all guaranteed to have the task of calculating its next iteration. The client has the task of executing the program, centralizing the global actions of the algorithm. More particularly, it is in charge of ordering the calculations and communications to be made, preparing the conditions for restarting the cycles as well as testing the convergence of the algorithm. The agent acts as a proxy between the client and the computing servers. This programming model is very well adapted to the case where targeted hardware architecture is a distributed system such as grid or cloud computing ones.

Algorithm 6 Parallel UCEL (in: LD, VD, n, ℓ, M, B^1, L^0, θ; out: B_{best}, L_{best}))

1: **Start.** Choose ℓ the number of the bags, M the size of the bags, L^0 the ℓ learners and

2: **Iterate.** For $i = 1, \cdots, \ell$ do in parallel
3: **Iterate.** For $j = 1, \cdots, q$ do
4: ***Training and testing - Client send training computations to servers***
Train L_i^{j-1} on B_i^j, produce L_i^j, test L_i^j on VD and select W_i^j.
5: ***Client receive training results and share informations***
share (B_i^j, L_i^j, W_i^j, AUC-score(L_i^j)) with all ℓ co-methods
6: ***Compute WVC and stopping test*** $WVC_j = V(L_i^j, AUC(L_i^j))$
$B_{best}^j, L_{best}^j, W_{best}^j = f(L_i^j, B_i^j, W_i^j, WVC_j)$
If (AUC-score(L_{best}^j) $> \theta$) then STOP all processes.
7: ***Sampling***
Set the bag $B_i^{j+1} = (1-\alpha) * W_{best}^j \cup (\alpha) * R_i^j$ where R_i^j is the set of $(m_i - k_i^j)$ correctly predicted data in B_{best}^j with $k_i^j = \mathbf{card}(W_{best}^j)$ and α is the updated weight given to miss-predicted data.
8: ***Result.***
Set L_{best} the best individual co-method or best weighted combination of co-methods during the iterations of all ℓ processes.

6.4 Synchronized direct communication parallel programming model

To study the performance of the behavior profiler proposed in section 5.8.1, we target a distributed memory parallel machine composed of nodes whose architecture can itself be parallel.

Hence for a second solution, to build a parallel synchronous UCEL approach, we choose a parallel programming model allowing direct communication between nodes. For the sake of simplicity and to have more determinism in the performance measures, here we opt for synchronous communications 6.1.

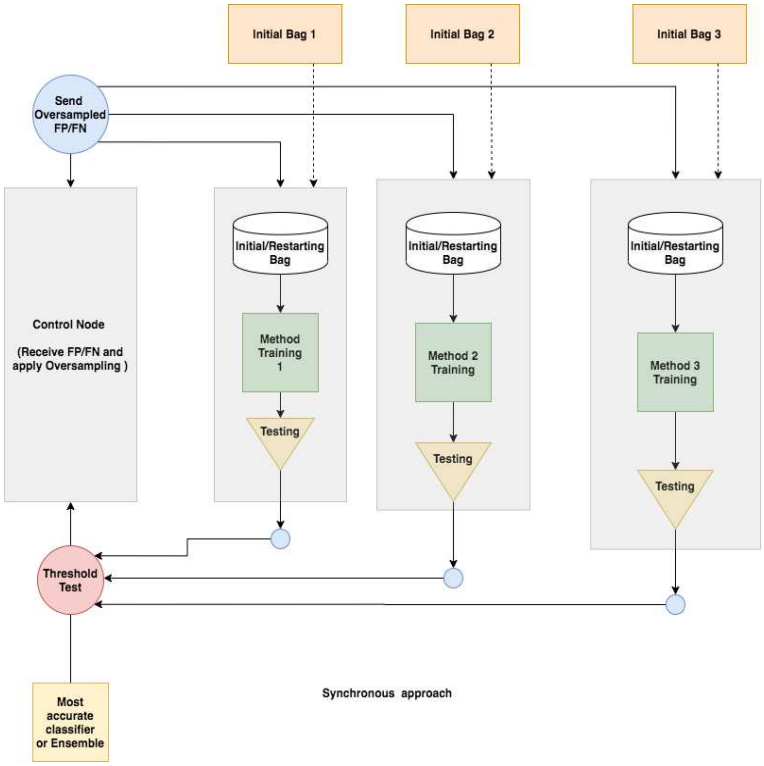

Figure 6.1: synchronous communication model

In this scheme, a co-method/process is assigned to a node, which will control it until the convergence of the UCEL algorithm. This means reaching convergence by this co-method or by any of the other co-methods. As with the targeted parallel programming model, it is possible to have direct communications between the nodes. The results from an iteration of a co-method can be sent to all the other nodes. In the same way, this node will receive the results of the other processes. Let ws be the quantity of data sent by node i to all of the other nodes at the end of an iteration and wr be the quantity of data received by node i from all of the other nodes after an iteration of their assigned co-method. Then, the quantity of data exchanged at the end of an iteration of each co-method is $(ws + wr) \times (\ell - 1)$ and the magnitude of data movement of the whole algorithm is upper bounded by $(ws + wr) \times (\ell - 1) \times q \times \ell$. One solution to minimize these communications is to emulate a shared memory in the targeted parallel distributed memory architecture.

This could be done by dedicating a node to the storage of the intermediate results of all nodes hosting a co-method as well as to the updating of the conditions for restarting the iterations of the co-methods by computing nodes, which we call the *control node*. Thus, each node sends the intermediate results of its assigned co-method to the control node ($\ell \times ws$) but will receive only the restating conditions rc to restart a new cycle of its co-method. The amount of data rc should be much less than wr. Consequently, the use of control node allows having $(ws+rc)\times q\times \ell^2$ as the upper bound of communication complexity of the whole algorithm.

Depending on the type and amount of information communicated between the computing nodes and the control

node, this model can be much less costly in the viewpoint of execution time or, in the worst case, as costly as when all communications are carried out directly between the computing nodes.

Let ℓ be the number of the bags (and co-methods or learners), m be their size, $B^1 = [B_1^1, \cdots, B_\ell^1]$ be the set of the initial bags, $L^0 = [L_1^0, \cdots, L_\ell^1]$ be a set of initial learners ($L^{-1} = 0$), W_i^j be the set of the false-positives and the false-negative resulting from the jth cycle of the ith co-method/classifier and, B_{best}^j be the training set with the best AUC-score among $B_1^j, \cdots, L_\ell^j$. This bag is associated with L_{best} the best learner (most accurate model) and W_{best} the lightest false positive-negative set (whose cardinal is the smallest). A co-method is considered as sufficiently trained if its AUC-score is larger than a classifier precision threshold. Let θ be this tolerance value. The parallel algorithm 7 realizes the proposed *behavior profiler* with SN$_i$ as ith server computing node and CN as a control node.

Algorithm 7 Parallel behavior profiler (in: TD, ℓ, q, θ, B^1; out: B_{best}, L_{best})

Start. Choose ℓ, m, B^1, L^0 the ℓ bags and learners, ...
Iterate. For $i = 1, \cdots, \ell$ do
  Iterate. For $j = 1, \cdots, q$
    Training and testing on SN$_i$
    Train L_i^{j-1} on B_i^j, produce L_i^j, test L_i^j on VD and select W_i^j.
    Communication: send from SN$_i$ to CN
    Send (B_i^j, L_i^j, W_i^j, AUC-score(L_i^j)) from CN$_i$ to SN.
    Computation and stopping test on CN
    $WVC_j = V(L_i^j, AUC(L_i^j))$
    $B_{best}^j, L_{best}^j, W_{best}^j = f(L_i^j, B_i^j, W_i^j, WVC_j)$
    If (AUC-score(L_{best}^j) $> \theta$) then STOP all processes.
    Communication: send from CN to SN$_i$
    Send $(B_{best}^j, L_{best}^j, W_{best}^j)$ to all node i for $i \in [1, \ell]$.
    Sampling on SN$_i$
    Set the bag $B_i^{j+1} = (1 - \alpha) * W_{best}^j \cup (\alpha) * R_i^j$ where R_i^j is the set of $(m_i - k_i^j)$ correctly predicted data in B_{best}^j with $k_i^j = \mathbf{card}(W_{best}^j)$ and α is the updated weight given to miss-predicted data.
    Result.
    Set L_{best} the best individual co-method or best weighted combination of co-methods during the iterations of all ℓ processes.

In the synchronous version of the algorithm, a server computing node trains a classifier allowing predicting classes and computing its AUC-score. At the end of this training and a testing cycle, a synchronization phase will take place. Then, all of the server computing nodes send a set of their results to the control node. This node is in charge of determining which is the best set among them before sending it to all the server computing nodes. This last sending requires another synchronization. This task of the control node between the two synchronizations can be seen as a critical section. Consequently, its execution time will have a significant impact on the execution speed of the whole algorithm.

6.5 Multiple user handling programming models

An insider threat detection module must be able to handle numerous users. With this consideration, we can use high performance computing techniques to deal with this problem.

For that purpose, we propose to use the instance of UCEL on different nodes. The global company dataset contains multiple users activities. We can give the specific user or a list of users to the same node. They then train the behavior profiler until their achievement of convergence. The nodes compute all iterations of the UCEL instances. In the end, each node can communicate its results to a chosen node to verify the performance and share them with a block of the insider threat detection module. The behavior profiles are stored in the file system.

6.6 Insider threat decision making module

For a global insider threat detection solution, we propose a cybersecurity module based on a parallel and scalable container architecture to detect insider threats. This software has the role of extending other cybersecurity tools with insider detection capabilities. It uses ensemble learning techniques and the UCEL framework ability to build employee behavior profiles.

The solution architecture is based on a component approach using an ensemble of containers. A container is a software unit that packages an application and all of its dependencies. Adopting a component approach allows to build independent and replace part of a global software solution. This allows to update the software capabilities easily. Since the component links with their dependencies, this approach also allows dealing with portability constraints.

The detection system is then an ensemble of containers mainly representing different parts of the machine learning pipeline and using the UCEL framework as its core.

6.6.1 Module abstraction

The global framework proposed will follow a model combining prepossessing, processing, and post-processing steps. The prepossessing part has a role in preparing data entries. They will be mainly a feed composed of users or role-associated activities. They are mostly activity datasets transformed in the format usable by analysis techniques (e.g., data frames).

The processing part will be centered around the notions of modeling a typical behavior profile for each user and the classification of their behavior. The establishment of this profile would depend on their past actions or their peer recorded activities.

We classify new user unlabeled data at the post-processing level using the behavior profile and establish an alarm system. This system has to give insight into the danger of some users actions. We essentially see this

approach as multiple boxes of methods able to individually give indications on the nature of the new conduct of a studied user. Their results are combined with having a stronger opinion on new unlabeled activity and taking contextual decisions (see 6.2).

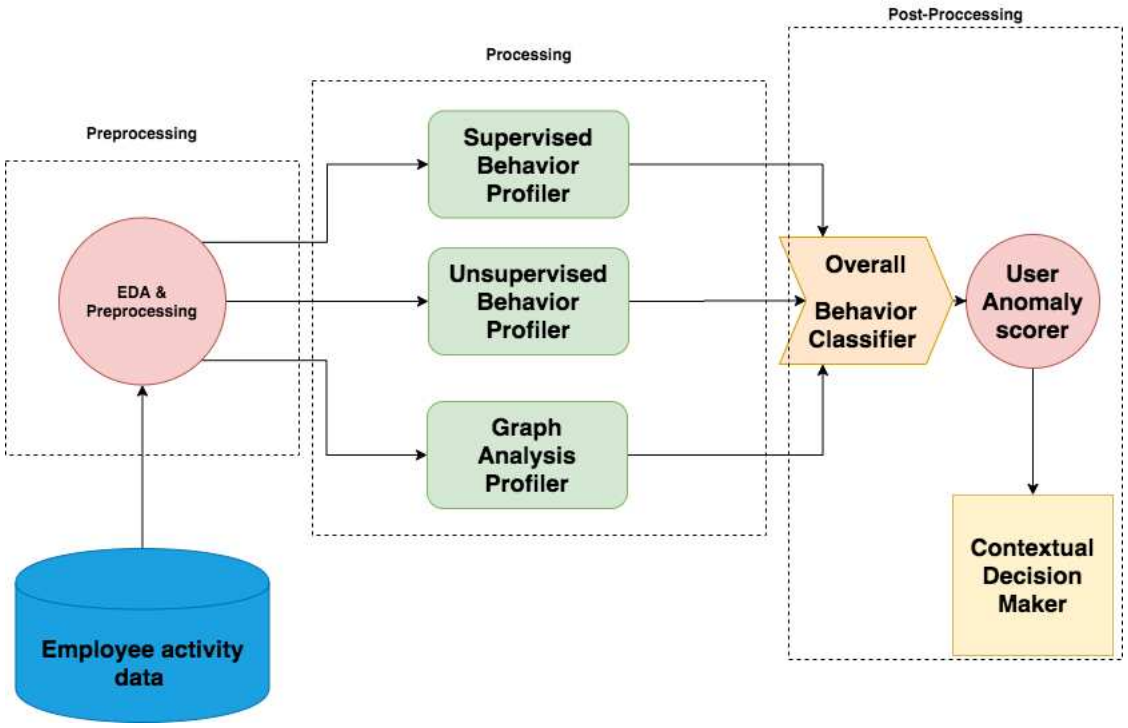

Figure 6.2: Framework abstraction model

6.6.2 Overall container-based architecture

To model this insider threat module, we propose an ensemble learning paradigm using docker containers 6.3. Each container is in charge of one of the blocks of the detection modules. The global insider detection modules follow an architecture model using machine learning pipeline steps such as exploratory data analysis (EDA), prepossessing, processing, and model serving as blocks (i.e., containers).

The prepossessing block has a role in preparing data entries. It uses a dataset feed from a data lake composed of users or role activities. The data is transformed in a format usable by data science techniques (e.g., data frames). This step use information from the exploratory data analysis block to do feature selection operations on the data and build a training, validation, and testing set. Additionally, the EDA block gives statistical insights on the data, clues on the type of methods we can use to detect insiders, visual indicators of attacks.

The processing block trains and validates numerous UCEL instances based on the same co-methods, anomaly detection methods, supervised learning methods, or independent classifiers. The trained model represents the behavior profiles. The establishment of this profile would depend on their past actions (i.e., user profiles) or their peer recorded activities (i.e., role profiles). This approach simultaneously offers multiple ways to detect if there is

an insider attack. These architectures also allow the integration of other machine learning frameworks such as the tree-based classification *(*XGBoost).

At the model serving block, we use an alarm system based on a behavior profile that classifies new unlabeled data and that establishes a user insider threat risk score. This score gives an indication of the danger user activities represent. These systems use the trained behavior profile and their AUC score to compute the risk score. Each trained profiler is used to analyze new unlabeled data from a systems bus receiving real-time activity data. We essentially see this scoring system as multiple methods that give indications individually on the nature of a monitored user.

This approach presents a significant potential for scalability and parallel computing on distributed systems. Consequently, this architecture is efficient in terms of time to solution. Container orchestrators such as *Kubernetes* and its machine learning toolkit *Kubeflow* can manage the scalability of this architecture and its industrialization.

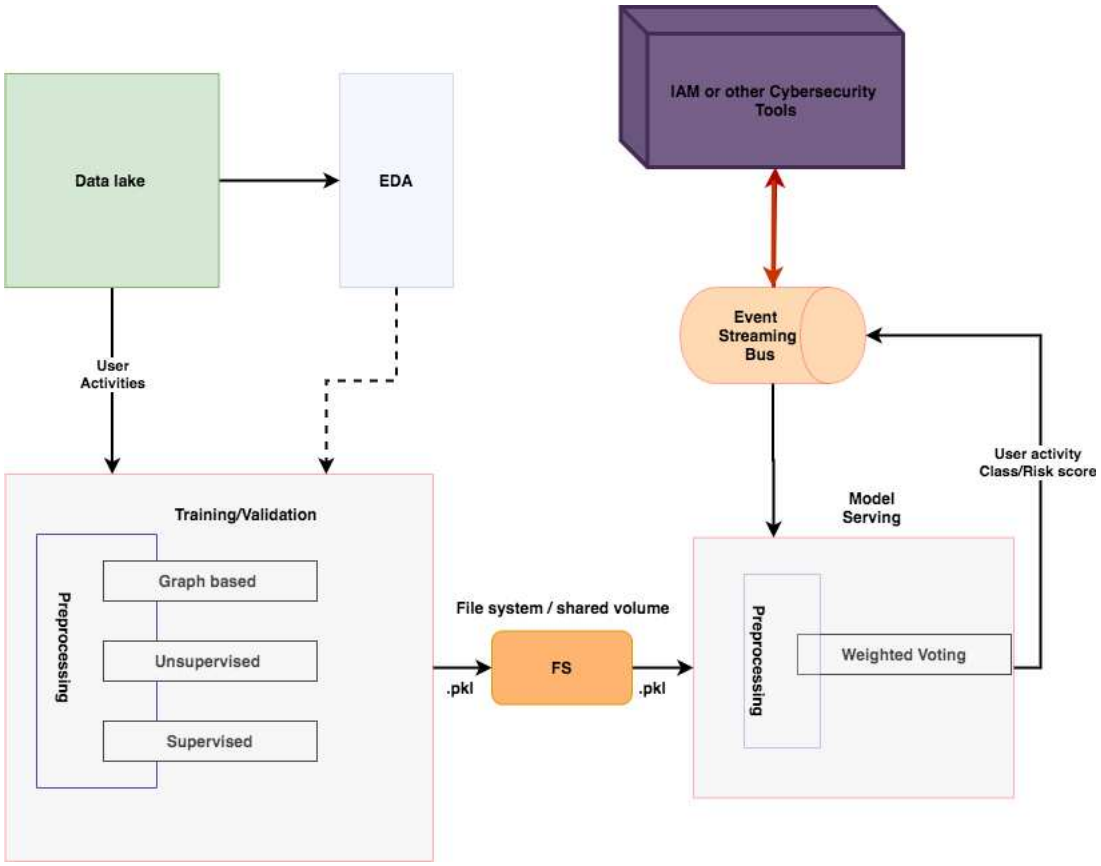

Figure 6.3: Container based framework

6.6.3 Employee risk scoring

Organizations can have numerous employees. Monitoring all of them at the same time is an almost impossible task. Cybersecurity tools usually use means in order to evaluate suspicious activities and treat them in order of priorities.

To evaluate the threat that can represent an employee to a company, we proposed to build an alarm system based on a risk score. This score is established using the classification result of the new unlabeled activities given to behavior profilers. This allows ranking employees in the function of threat suspicions.

To goal of the alarm system is to compute a risk score associate with a user during the day. This score is sent to a security administrator, a security operation center (SOC), or contextual decision systems through the systems bus. This information can also be used by other cybersecurity tools such as SIEM and CASB. It is essential to note that risk scores are sent in case they reach fixed thresholds. Depending on their values, we can decide to send an alarm or to take a contextual decision. To stop the attackers as soon as possible, they are established in real-time or daily. This also allows building daily activity reports.

We built an alarm system using the election and voting principles of ensemble learning techniques. Each time a user event occurred and was sent in the bus, the ensemble of behavior profiles gives their class prediction with their previous AUC computed during the training and the validation phases. If the behavior profile spots an abnormal activity, the base score is increased. The amount of this raise depends on the risk scoring techniques use. The risk score is stored to be potentially aggregated.

We name the technique used anomaly scorers (AS). For now, we propose two different anomaly scorers that we present in more detail in the validation part of this thesis III. These AS are based on the choice of the best detection technique or a weighted linear combination of classification results and their performance measures.

Hence, with this system, we initially attribute a base score for each user. In our case, we choose a base score equal to *50*, but this choice was made arbitrary. We add an alarm threshold fixed at *55* and a contextual decision action threshold at *60*. The thresholds are chosen to have specific responses depending on the score. These responses are mainly the launch of an alarm or contextual grey decisions.

This double level of thresholds with grey decisions is more suitable to an insider threat problem since they have the tendency to generate false alarms. Black and white decision are more binary responses, such as allowing or denying a user access to the network or company endpoints in the function of their score. They can impact the employees productivity if there is a false alarm.

Additionally, to classify employees behavior visually, we associated a color code to risk score each time the threshold is passed. Green would represent a clean user, orange a suspicious user, and red a confirmed malicious user (see figure 6.4).

6.7 Conclusion

In this chapter, we presented the intrinsic parallelism of the UCEL framework and proposed a couple of programming models and implementation specificities to exploit the parallel properties of UCEL and the requirement of the insider threat detection problem. We also presented the integration of the UCEL framework to a global insider detection

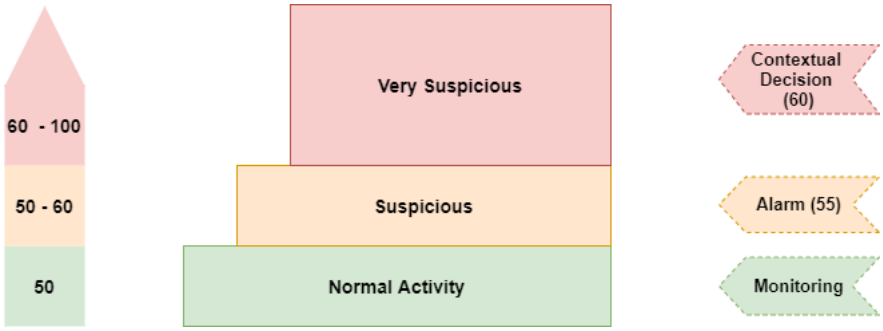

Figure 6.4: Risk scoring choices

module. Additionally to the daily and alarm systems, it is also essential to monitor users in a more significant period (i.e., week, month). This might be important to spot an attacker that operates slower and would spread its malicious actions in a larger time-space. A risk score based on weeks and months allows building activity reports off longer periods. They can be sent to security specialists. They can also be an efficient approach to detect insiders.

Part III

Framework validation by experimentation

Chapter 7

Experiments context

7.1 Introduction

We start by defining the experiments conditions, the datasets we used and the performance metrics we chose. Since our investigation centers around building an insider threat solution in the general sense, the majority of experiments we have done is on a UEBA system. However, we started by focusing on IAM-based Insider threat detection.

The UCEL approach we proposed is focused on the general approach since the IAM data we dispose is limited. We also used datasets from other domains to illustrate the overall performance of the UCEL framework. The first UCEL experiments are following the sequential algorithms we proposed. We focus then on the parallel tests. We analyze every experiment and give corresponding conclusions and perspectives. Hence, we start by describing the dataset, experiment tools and experiments conditions.

7.2 Dataset of experiments

We firstly experimented with two unlabeled simulated audits IAM dataset, one from an endpoint access management (EAM) software and the second form an identity and governance application. These data are from an enterprise that produces IAM management software. However, since they are audit data, they are really sparse. These audit data are only created when administration systems interrogate the systems to get specific information on the user, but are not necessarily complete information to investigate behavior. The fact that they are created on-demand makes it challenging to track the evolution of the features, hence user activities over time.

For the second series of experiments, we use the open-source version *CERT: Insider threat R4.2* first and then*R6.2*. As we stated before, the *Computer Emergency Response team* (CERT) dataset is an artificial insider threat dataset created by the *National center for internal threats* (NITC) division. It is a set of data composed of employees regular post-login activity in a simulated company environment context and insider attack scenarios per-

petrated by synthetic malicious actors. These scenarios are abnormal and suspicious activities that are dangerous for enterprises. This dataset is the most used in the domain of insider threat detection. The *R4.2* version is the dense needle version containing multiple users for the first three scenarios. However, even if there is only one user by attack scenario, the *R6.2* version is more recent and contains more scenarios (5 scenarios).

The UCEL behavior profilers we built, learn user behaviors using the training set and the incorrectly classified elements in the validation set from the *R4.2* and *R6.2* datasets. Before that, we applied different prepossessing steps to prepare the data for the models. We also test the UCEL framework with datasets in other domains that we describe in Chapter 10.

Hence, to show the reliability of the *BP*, we considered the following insider threat attacks scenarios. We specially focus on scenarios 2 and 3. More information about this data can be found in [35]) :

1. User who did not previously use removable drives or work after hours begins logging in after these hours, using a removable drive, and uploading data to wikileaks.org. He leaves the organization shortly thereafter.

2. User begins surfing job websites and soliciting employment from a competitor. Before leaving the company, she/he uses a thumb drive (at markedly higher rates than their previous activity) to steal data.

3. System administrator becomes disgruntled. Downloads a keylogger and uses a thumb drive to transfer it to her/his supervisor's machine. The next day, she/he uses the collected keylogs to log in as his supervisor and send out an alarming mass email, causing panic in the organization. She/he leaves the organization immediately.

4. A user logs into another user's machine and searches for interesting files, emailing to their home email. This behavior occurs more and more frequently over 3 months.

5. A member of a group decimated by layoffs uploads documents to Dropbox, planning to use them for personal gain.

7.3 Hardware architectures

We start by serial programming using a MacBook with an Intel 'i7' *Haswell* processor with 4 core clocked at 2GHZ, and 8 gigabyte of memory. However using this hardware limits the quantity of data we can train (i.e., max 100000 entries). We then switch to the supercomputers we had the access to. The parallel architectures used as support for the presented experiments is Grid'5000 (G5K). It is a french national and international cluster constituted by several large homogeneous sub-clusters situated through France, Brazil, and Luxembourg. This large-scale and flexible platform is a test-bed for experiment-driven research in all areas of computer science. It focuses on parallel and

distributed computing. G5K resources are 15000 cores, 800 computer nodes, and also technologies such as GPU, SSD, and Infiniband. For our experiments, we used mainly nodes from the cluster of the Lille site.

We also used an enterprise cluster with virtual machine installation of docker and docker-compose to create the necessary images, containers, and machine learning pipelines for the global insider threat detection module. Docker-compose allowed us to launch at the same time the data-lake, the preprocessing, and processing containers.

7.4 Performance metrics

7.4.1 Machine learning methods performance metrics

At the begging, we use different classification performance metrics, such as *precision* 7.4.1, *recall* 7.4.1, and *F1-score* 7.4.1.

$$Precision = \frac{TP}{TP + FP} \tag{7.1}$$

$$Rappel = \frac{TP}{TP + FN} \tag{7.2}$$

$$F1 - Score = \frac{precision.recall}{precision + recall} \tag{7.3}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{7.4}$$

However, we finish by settling with the AUC-score that is more suitable for the unbalanced dataset that we find on the insider threat detection problems. The AUC is the area under the receiver operating characteristic (ROC) curve. This curve represents the performance of a classification model for all classification thresholds and plots the rate

of true positives $\frac{TP}{TP+FP}$ against the rate of false positives $\frac{FP}{FP+TN}$ where the letters T, F, P, N denote true, false, positive and negative. By calculating AUC, we get a performance measure for all possible classification thresholds.

A model whose predictions are 100% wrong has an AUC of 0.0, and the one whose predictions are 100% correct has an AUC of 1.0. AUC is a desirable metric because it is scale-invariant and classification-threshold-invariant. This metric allows modification of the classification threshold for a more restrictive insider detection without having an inaccurate performance measure. Also, it performs well with a highly imbalanced dataset.

Hence to evaluate the performances of our behavior profiler algorithms 5.6, we use the AUC-score allowing measurement of prediction performance and, consequently, (in)validating the approach. Indeed, from this information, we can see if the BP algorithm improves the classical insider detection methods in terms of the FP/FN rate reduction and handling of the bias and variance tradeoff.

7.4.2 Parallel models performance metrics

The second metric concerns evaluation of performances of the proposed implementation of parallel models in the point of view of computation speedup. The speedup denotes the gain of time obtained by the exploitation of the parallelism of the BP algorithm. More precisely, the speedup can be defined by the ratio of the execution time of a sequential algorithm to the execution time of the parallel version of that algorithm to solve the same problem. We also measure the performance of our parallel behavior profilers using scalability curves. These curves help determine if there is a positive impact on computation performance when we increase computing cores or the size. We gives more details about the experiments conditions in 11.2.

7.5 Software implementation tools

The implementation of this UCEL framework leveraged the use of *python* code. We chose *Python* because its commonly used in the data science domain. Libraries such as *pandas*, *scikit-learn*, *tensorflow* are very use full to build machine learning-based solution. For graph analysis the *networkx* an *scikit-network* offer means to visualize and compute features. A combination of these libraries allows us to build the UCEL framework and the machine learning pipeline in the insider threat detection module. The *pandas* library functions will allow us to manipulate multiple type of activity data. The raw datasets would much likely be under databases format extraction (i.e., .xls, .xml or .csv files). The exploratory data science and feature selection techniques in the exploratory data science block used *pandas*, but also *matplolib*, *seaborn* for data visualization and *scikit-learm* for data sampling and splitting. For the oversampling phase in UCEL we also use the *imblearn* library.

Custom implementation of the PageRank and autoencoder-based methods are based the oriented object programming and the *scikit-learn*, *tensorflow* libraries. We create new classifier classes, with *fit* and *predict* methods.

These libraries can be used to apply classifiers to activity data. These libraries are widely used in machine learning communities, but a custom implementation is more suitable to build optimized software.

To implement the parallel programming model, we mainly used the message passing interface (MPI), *mutithreading*, *multiprocessing* and *joblib* python libraries. The MPI (*mpi4py*) library can be used to do distributed calculations on the ensemble learning model and the eigenvalues problems. The *mutithreading*, *multiprocessing*, the *joblib* backend are essentially libraries that deal with multithreading with some specificities. They share the working function on the available core resources.

The autoencoder method is based on the use of a neural network. We proposed an efficient implementation capable of using GPU. The iterations of the methods are sent to the GPU when they are available with the *CUDNN* library.

We also proposed hybrid parallel programming models that combining *MPI* and *joblib* backend. We run tests on a cluster architecture (see 11), where the nodes are in charge of a single user, and the threads would manage the distributed part of the base of the UCEL framework. We give details about the performances of these implementations in the validation part of this thesis III.

7.6 Conclusion

The experiment tools we use are mainly the different datasets to test the proposed solutions and the metrics we used. We also describe the libraries we used to test the framework. The conditions are the material and software structure user to test the implementation of our codes. For now, the solution we offer is yet to be industrialized. However, in our general conclusion, we share insight into the industrialization process. In the next chapters of this part, we present the preprocessing method we use. Following this part, we present the experiment on IAM-related scenarios, the tests to validate UCEL and the overall detection models, and we finish with the parallel test we propose.

Chapter 8

IAM based insider threat experiments

8.1 Introduction

In this chapter, we present the first tests we did at the beginning of this thesis. We used classic classification, anomaly detection methods, and graph-based methods on two IAM datasets. This part has the goal to give insights on the accuracy of a list of machine learning methods on an IAM dataset before choosing to use them for an insider threat detection solution. One of the goals of this work is to build a UEBA extension module for an IAM software suite. The study of this data and test of machine learning solution is useful to build a future complete solution.

8.2 IAM audit data and experimentation description

We dispose of a total of two proprietary IAM datasets: two IGA datasets. As we discussed before since the data is recovered from an audit system, its constitution very sparse. The user behavior associated with this data is then incomplete. Besides, these data are from simulations made by the IAM software companies. These have the roles of setting examples of data generation and not specifically simulate behavior. At this moment, we didn't use particular preprocessing techniques on this data. We are mainly suppressing the corresponding incomplete line and choose the feature using expert knowledge. We then encode the data from categorical to a numerical value.

The first IGA dataset is made of permission request events. The second IGA dataset is a policy dataset made of but has only five features. And policy data represents the list of roles and permission of the users(See Table 8.1).

In the first experiment, we used unsupervised hierarchical clustering on the IGA permission request and policy data to determine if we can cluster users or roles in the function of the permission request rights. As we discussed in the related work, the IGA data can detect a high privilege access abuse attack. For that goal, we started with a role classification in the function of the permission assignment. We also wanted to check if the clustering process could highlight outliers in the data. Hence, we also applied an IForest method to the policy data.

Table 8.1: IGA dataset presentation

Data	Features	Size (# events)
IGA Permission Request	'Status', 'Authorization start', 'Authorization end', 'User', 'Role assignment start', 'Role assignment end', 'Business rule', 'Organization', 'Permission assignment start', 'Permission assignment end', 'Permission assignment role', 'Permission assignment organization', 'Permission', 'Permission group', 'Application', 'Resource', 'Resource group', 'Operation', 'Operation group', 'Service', and 'Service group'	14720
IGA Policy Data	'User Id', 'Organization code', 'Organization inheritance', 'Role and Role ID', 'Permission' and 'Permission ID'	51

As the second experimentation, we use once again the IGA users access right to request and anomaly detection methods to determine if they request are anomalous. We mainly use the IGA data and extract a one-user full rights request list that represents the expected behavior and small subsets of others user data that are supposed to represent the outliers of our experimentation. The goal of this experimentation is a user reconnaissance method that is fairly close to the detection behavior anomalies.

In the third experiment, we use the supervised method. The goal was to determine if permission can allow recognizing roles using several classification methods. This was also important because we wanted to build a role-based behavior profile.

At last, we did graph-based visualization experimentation on the IGA data based in order to check the possibilities of graph feature analysis.

8.3 First experiment: user role clustering

8.3.1 Clustering on the permission request data

In this experiment, our goal was to check if it is possible to distinguish different user roles using the IGA data. We started with the permission request dataset since it has more entries. We built a subset of 531 entries compose of four different user roles, which are different administrators.

For that purpose, we applied a principal component analysis (PCA) to compress the data but still keeping its properties and reduce the number of features. It also allows a representation and a visualization of the data in a 3D format (see Figure 8.1).

Here we applied the mean-shift (MS) algorithm. This is a centroid-based algorithm that detects the number of clusters in data. It is based on locating the maxima of a density function. This maximum is the best center of a cluster.

If we analyze the Figure 8.1, the MS algorithm could distinguish between the four administrators by detecting

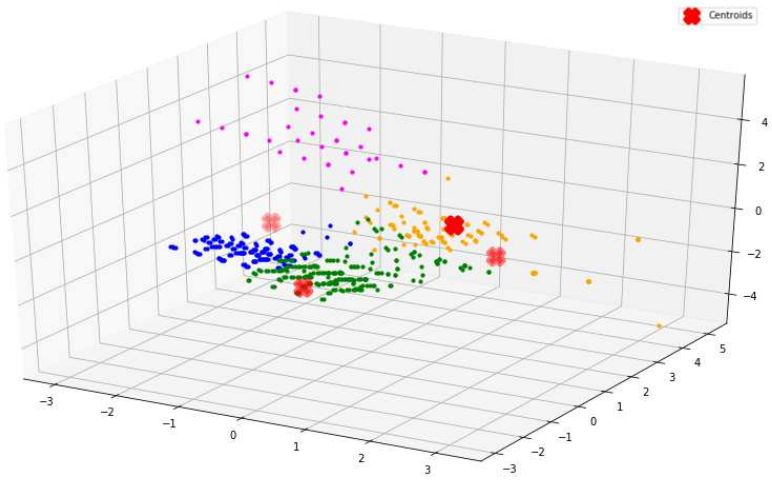

Figure 8.1: Mean shift on permission request dataset PCA 3D

four different clusters. This first experiment was a success. If we investigate the data, we remark that all four administrators are distinct permission feature values, which probably facilitate the task of the methods.

8.3.2 Outliers detection on the policy datasets

We applied the same experiment based on the MS algorithm on the policy data (see Figure 8.2). However, this time with the full data (51 entries). In this case, we found two clusters, and one clear outsider was found at the edge of the domain. If we look into the data, this user has not the same organizational inheritance as the rest of the users, which are all sensibly in the same area. The organization inheritance means that the user does inherit the permissions linked to his organization. This implies that the mean shift algorithm recognizes a user policy with a different organization.

Since we remark an outlier on the policy data experiments, we tried to use an anomalies detection method to check if it could detect it. Hence, we applied on the same dataset an isolation forest algorithm (see Figure 8.3). We found a list of outliers, which also contain the same individuals at the edge of the domain.

In a way, this allowed us to confirm the result of the clustering algorithm and showcase one of the usefulness of the PCA transform to detect user outliers.

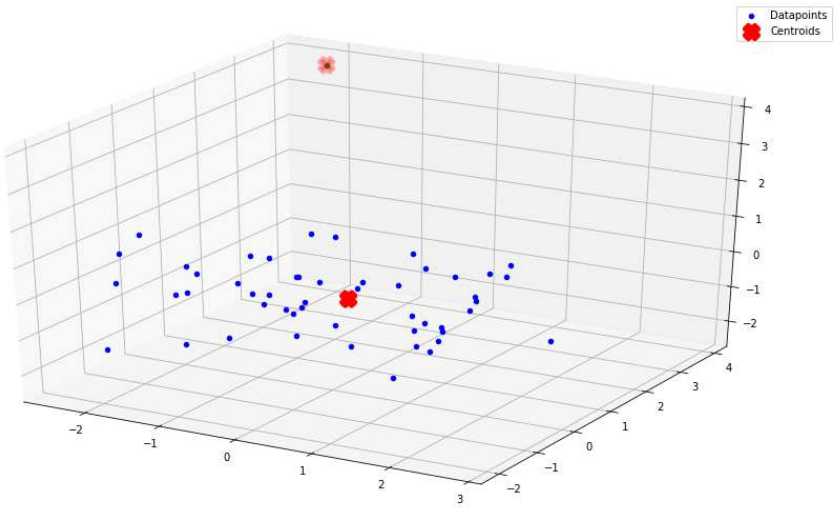

Figure 8.2: Mean shift on policy dataset PCA 3D

8.4 Second experimentation: user request anomaly detection

For a second IAM experiment, we worked on a subset of the permission request dataset of 364 entries, composed by 251 entries for *User1*, 30 entries for *User2*, 10 entries for *User3* and 10 entries for *User4*. Our goal was to simulate a scenario where *User2*, *User3* and *User4* entries would be idealy detected as outliers of this subset.

At first, we applied to our dataset four outliers detection methods, which are an OCSVM, an IForest, a robust covariance (Robcov) (i.e., Elliptic envelope), and the local outlier factor (LOF) methods. We then use a voting classifier combining the base-classifiers, hoping to get a better result. We finally present the results in the Figure 8.4 and the Table 8.2.

Table 8.2: Anomaly detection methods metrics

Classifers(%)	Precision	Recall	F1-score	AUC-score
IForest	0.73	0.83	0.75	0.83
OCSVM	0.65	0.66	0.66	0.66
LOF	0.63	0.61	0.62	0.61
Robcov	0.93	0.86	0.89	0.86
ODVtg	0.81	0.88	0.83	0.88

It is important to note that changing the studied subset (i.e., choosing different users in the subset) impacts the accuracy of the detection methods, but overall in our test, the Robcov and IForest consistently outperform the other algorithms.

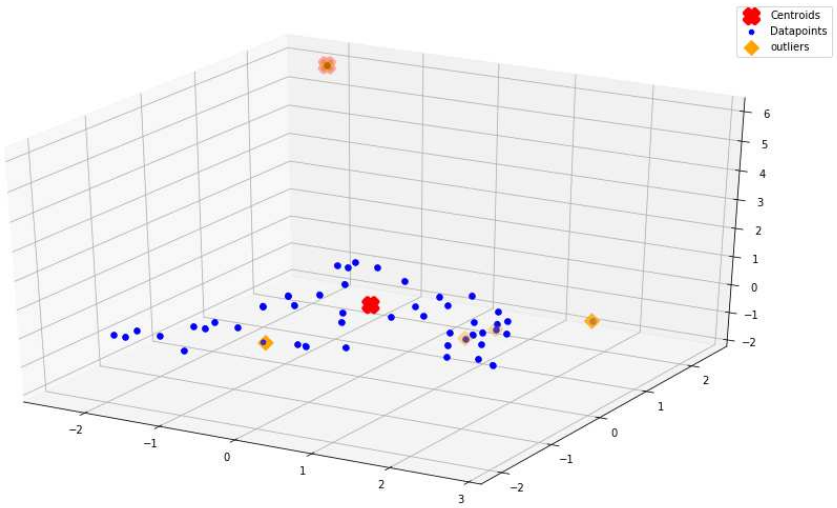

Figure 8.3: Isolation forest on policy dataset

8.5 Third experimentation: supervised learning

In the third experiment, we test the utility of the supervised learning methods for role classification in the function of the permission request dataset. Like the previous experiment, this test can be useful to detect high privilege access abuses. With this experiment, we are looking for the performance of supervised classification methods with the IAM dataset. The main goal was to determine if the supervised classification approaches can distinguish user roles in the function of their permission request. Here we also used a sample of the permission request dataset, using random sampling without replacement. There are 38 roles in the dataset subset of 11380 entries we use for this test.

We use classic supervised learning techniques such as logistic regression (logreg), decisions trees (DT), K-Nearest neighbors (KNN), linear discriminant analysis (LDA), gaussian naive bayes (GNB), support vector classifier (SVC), random forest (RF), multilayer perceptron (MLP), bagging Classifier (BC), adaptive boosting (ADB) and gradient boosting classifier (GBC). The ensemble learning is using decision trees as base classifiers.

The classification accuracy using this dataset is presented in Figure 8.5.

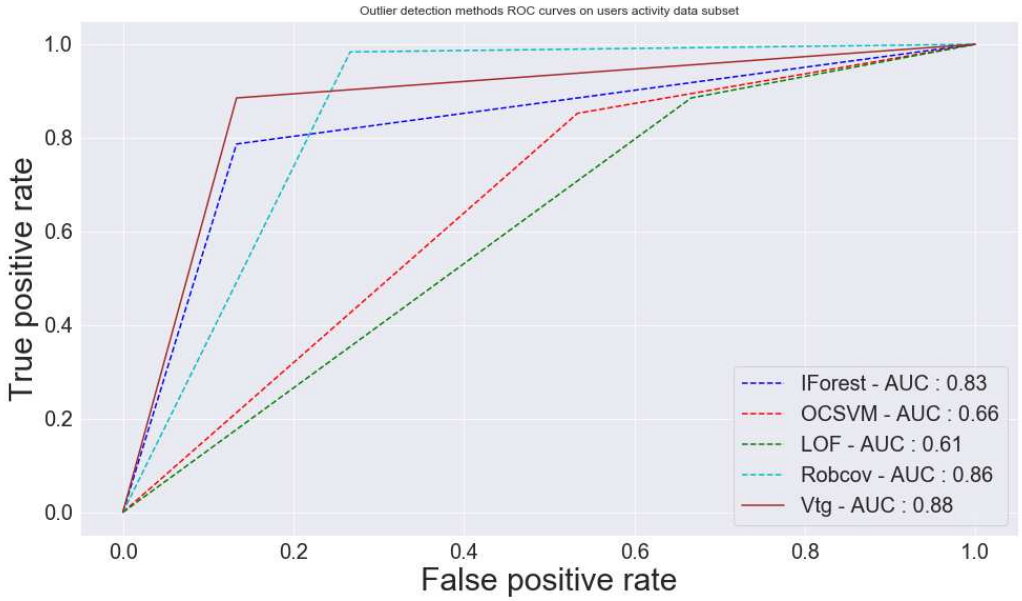

Figure 8.4: Receiver operating characteristic curve for outliers detection methods

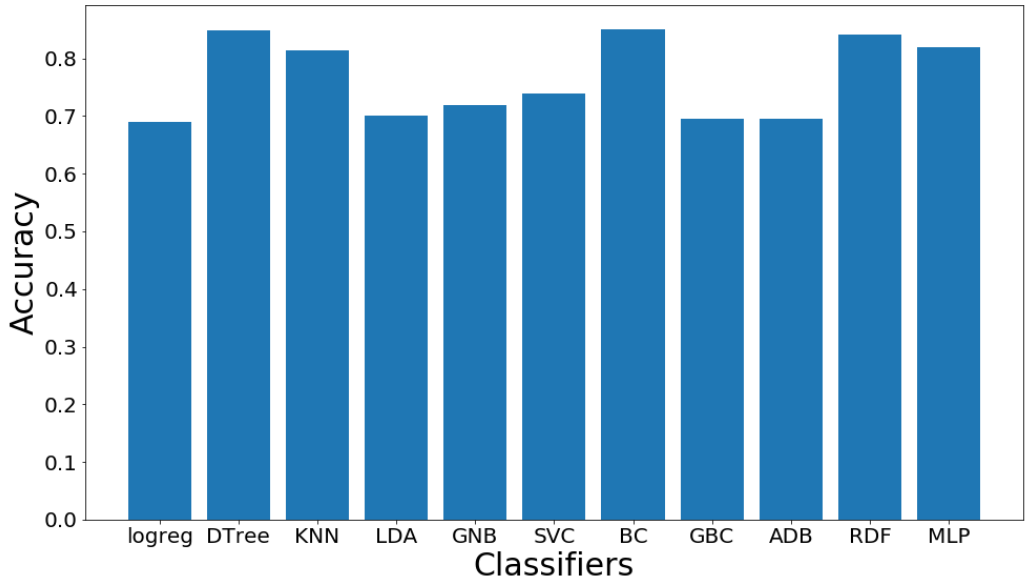

Figure 8.5: User classification recall

With this experimentation, we use grid search cross-validation to get the best hyper-parameters for each classifier. Given the result in Figure 8.5, we found out that this dataset allows classifying users roles up to 85% of accuracy, which is pretty accurate. The best methods are the method that uses ensemble learning, such as bagging, boosting, and the MLP. These results brought our attention to techniques to solve our insider threat problem

more.

8.6 Graph-based IAM data visualization

Our last IAM experiment was to convert the dataset at our disposal under the from a graph. This allowed us to visualize the data into a new format and studied its ability to detect insider threats. In the chapter 2, we found techniques dealing with insider threat and fraud detection using graph representations analysis techniques and linear algebra. Data samples can be represented using components such as nodes and edges. The study graph features can be a critical source of information to detect anomalies.

One of the objectives of this thesis was to build a solution to the insider threat problem. Initially, we were planning to create a solution based on the combination of graph-based belonging to the domain of, *Subgraph Analysis*, *Propagation Methods* and *Latent factor models*). In the end, we mainly use PageRank Based Methods.

Our goal was mostly to identify the nature of the nodes and edges in a graph representing IAM data in this experimentation. Common nodes and edges feature can represent normal behavior, and the anomalous ones, abnormal behavior. Strictly following a role-based access control paradigm, a possible representation of the permission request data would be close to the representation proposed in Figure 8.6. In this figure, we represent a small data subset to check the relation between the different features of the data. Organizations are represented in green, users in blue cyan, roles in light blue, permissions in magenta, applications in purple, and operations in grey.

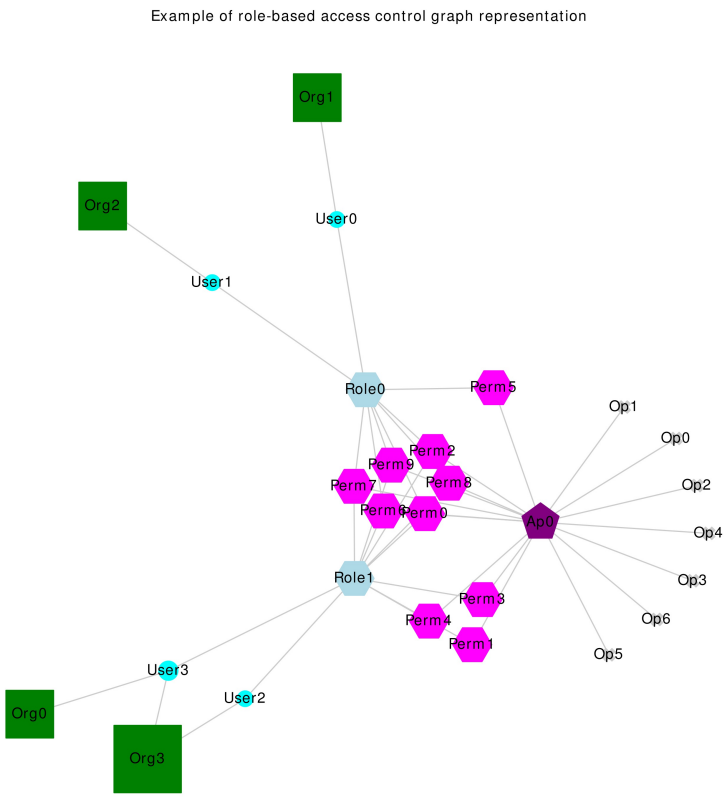

Figure 8.6: Graph modelization of a IAM permission request dataset

We can observe in this representation that the permissions are linked to multiple roles. The roles are linked to users, which are linked to one or multiple organizations. The permission gives access to applications and allows operations in these applications. This implies representation allows us to see what are the important variable for roles and user classifications. It also highlights that users in different organizations can share permission and access features of the same application. This representation allows learning a couple of things on IAM data. However, we keep in mind that the test was done on a small subset. Hence more relation could be found using other subsets.

8.7 Conclusion

Since one of the goals of this work is to build an extension of the IAM software suite, in this chapter, we analyzed the IAM data at our disposal. We pointed out the detection methods based on graph modeling, supervised and unsupervised learning methods applied to IAM data can highlight some characteristics of the data. Overall, we

obtained the best accuracy results using supervised methods, and ensemble learning methods (85%) followed closely by tree-based (84%) methods and an artificial neural network for our second test (82%). This result on the IAM data are interesting, but the tests were limited by the data capability to deal with insider threat in general. Hence in the next chapter, we use the more complete CERT data as the basis of our experimentation.

Chapter 9

CERT data exploration and preprocessing

9.1 Introduction

In this chapter, we focus on a more general aspect of insider threat detection. We have discussed in 2 that we are limited in the number of scenarios we can process using the IAM dataset. That dataset is also based on audit data, make it challenging to track a user behavior evolution over time. Hence in the following sections, we focus on the data analysis and the preprocessing of the CERT dataset and the associate insider threat attack scenario. This dataset contains more instances, and it not sparse at the opposite of the IAM audit dataset we treated in the chapter8. This implies a better representation of user behavior. We primarily focus on the two attack scenarios, which are close to the IAM context however more general.

Hence, we present here an exploratory data analysis, the review, and the preprocessing of the CERT data. This allows choosing the features for the training and validation of machine learning methods permitting to obtain good classification results.

9.2 Data exploration of CERT

9.2.1 Data features

As we presented before, the CERT Data has multiple versions. We mainly focus on two versions: the dense needle *R4.2*, and the most up-to-date *R6.2*. These two versions are mainly composed of several user activity data files. The *R6.2* version also has an additional files titled *decoy_files.csv*. We introduced the principles of decoy files in the chapter 2. The normal activities of each user are spread in these main (.csv) files, except for the LDAP.csv, which contains information on the user identity. We present their features in Table 9.1 and 9.2.

All the activities from the main CERT files have a timestamp, an individual identification number, and the *User*

Table 9.1: CERT dataset presentation *R4.2*

Data	Feature	size (events #)
email.csv	'id', 'date', 'user', 'pc', 'to', 'cc', 'bcc', 'from', 'size', 'attachments', 'content'	2629979
http.csv	'id', 'date', 'user', 'pc', 'url', 'content'	28434423
file.csv	'id', 'date', 'user', 'pc', 'filename', 'activity','to_removable_media', 'from_removable_media', 'content'	445581
logon.csv	'id', 'date', 'user', 'pc', 'activity'	854859
device.csv	'id', 'date', 'user', 'pc', 'activity'	405380
pyschometric.csv	'employee_name', 'user_id', 'O', 'C', 'E', 'A', 'N'	1000
LDAP	'employee_name', 'user_id', 'email', 'role'	1000
*AttackScenarios.csv	'source', 'id', 'date', 'user', 'pc', 'to', 'cc', 'bcc', 'from', 'activity', 'size', 'attachments', 'content'	*Depends on the scenario

Table 9.2: CERT Dataset Presentation *R6.2*

Data	Feature	size (events #)
email.csv	'id', 'date', 'user', 'pc', 'to', 'cc', 'bcc', 'from', 'activity', 'size', 'attachments', 'object', 'content'	10994957
http.csv	'id', 'date', 'user', 'pc', 'url', 'activity', 'content'	117025216
file.csv	'id', 'date', 'user', 'pc', 'filename', 'activity','to_removable_media', 'from_removable_media', 'content'	2014883
logon.csv	'id', 'date', 'user', 'pc', 'activity'	3530285
device.csv	'id', 'date', 'user', 'pc', 'file_tree', 'activity'	1551828
pyschometric.csv	'employee_name', 'user_id', 'O', 'C', 'E', 'A', 'N'	4000
decoy_file.csv	'decoy_filename', 'pc'	31095
LDAP	'employee_name', 'user_id', 'email', 'role', 'business_unit', 'functional_unit', 'department', 'team', 'supervisor'	6356
*AttackScenarios.csv	'source', 'id', 'date', 'user', 'pc', 'to', 'cc', 'bcc', 'from', 'activity', 'size', 'attachments', 'content'	*Depends on the scenario

ID they associated. They can considered as events since they are linked to a timestamp. This feature allows for tracking the evolution of activities through time. They represent the activity of a user. These files mostly contain categorical data. It is important to note that we added a *'source'* feature to all the datasets to have a piece of information on the type of data that a sample belongs to.

The *R4.2* is the denser in terms of examples of attacks. For each of the attack scenarios presented previously, there are multiple malicious users hence multiple **AttackScenarios.csv*. These files are made of user malicious action events. Depending on the scenario, they can include suspicious email exchanges, strange online behavior, and usage of the external device at strange hours. However, even if there is only one malicious user by attack scenarios, *R6.2* has more features that help to characterize user activities. Hence, *R6.2* represents more accurately a behavior. In our experiments, we use the *R4.2* briefly before switching to the scenarios 2 and 3 of the R6.2 version.

9.2.2 CERT general analysis

The behavior profiler of the UCEL framework builds user or role profiles. They allow to classify new unlabeled activities as normal, abnormal, and if they are insider attacks. We need to extract all event links to the same user or to an ensemble of users with the same roles to use this approach. Next, we merge and sort the data in the function of their timestamp. Hence, we create subsets for the versions *R4.2* (for scenario 2), R6.2 (for scenarios 2 and 3).

Since we know the available source of normal activities and the attack scenario, we labeled the data accordingly. We use binary labeling, 0 for normal activity and 1 for abnormal.

At this level, our goal is to observe and check if we can get valuable information on the dataset. We looked for some factors that we can visualize and some patterns indicative of insider attacks. Hence, after the data merge, we used individual data to analyze its features.

The figures 9.1, 9.2, and 9.3 present respectively histograms of the merge data file sources for the subset of the *R4.2*, *R6.2* second attack scenario and the *R6.3* third attack scenario.

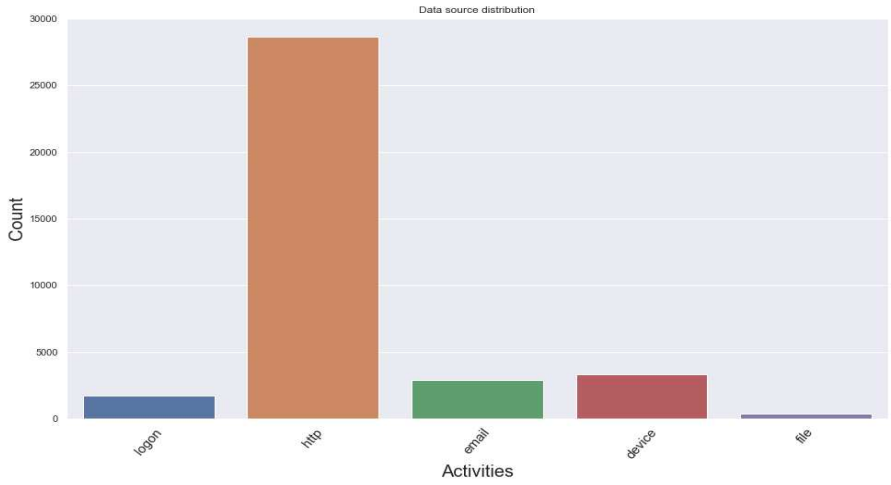

Figure 9.1: Source of the data for the R4.2, scenario 2

These figures give information on the sources of events and the data volumes in the function of the version and the scenarios. For all the users and roles studied, the event from the HTTP files is the most common. Hence users mostly do actions link to the web. This can be the access to the internet site, download or upload information.

The figures 9.4, 9.5 and 9.6 present histograms of the spread of the activities in function of the month and the year.

The figures on the monthly activities give us insight into different types of activities and their monthly volumes. This showcases that the *R4.2* lacks activity datasets. We chose the string 'empty' to replace the blank data entries after the merging of the files. Like we discussed before, the *R4.2* is not rich in diversity of activities compared to the *R6.2*.The goal here was to see if there is a change in the activity volumes in the function of the month. A

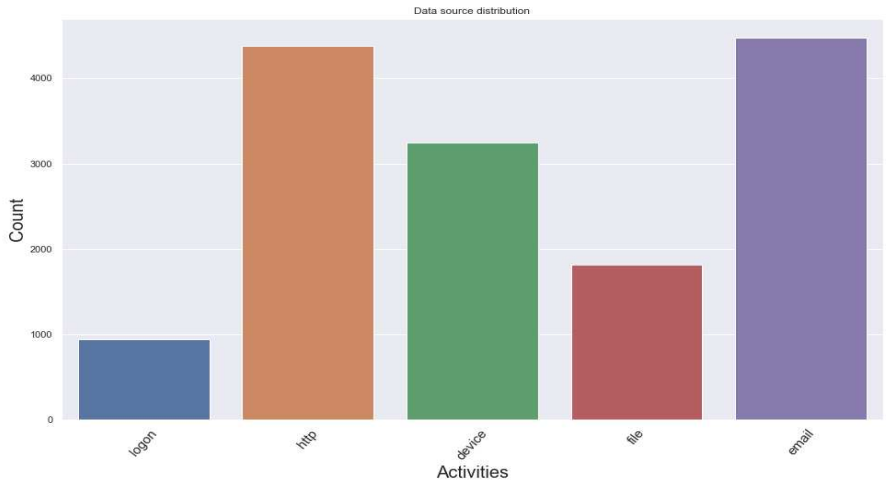

Figure 9.2: Source of the data for the R6.2, scenario 2

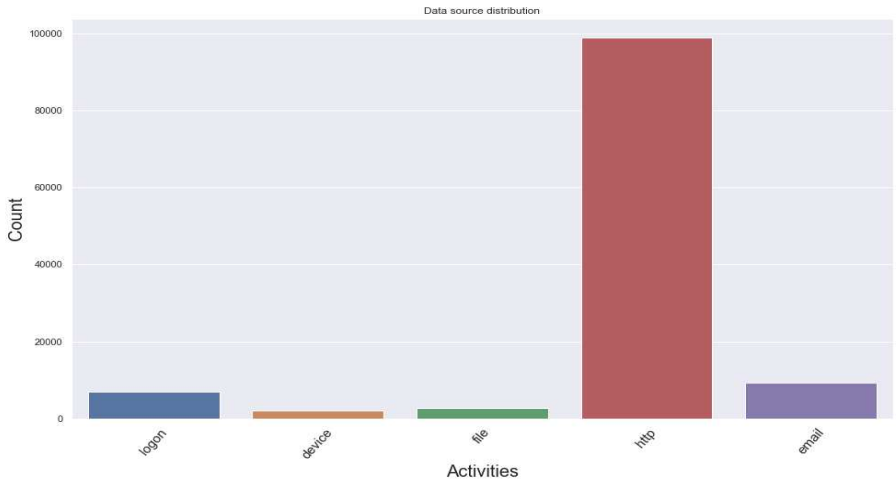

Figure 9.3: Source of the data for the R6.2, scenario 3

volume change could be indicative of habit change or an insider attack. Here we choose a month to have more of a macroscopic view. However, in the following sections, we propose daily activities. With these figures 9.4, 9.5 and 9.6, we didn't remark clear and particular changes of behavior. However, with the daily activity, we find some interesting variations.

The figure 9.7 and the 9.8 present a Pearson correlation (PC) in form of a Heat-map of the different data features. The PC coefficient measures a linear correlation between to set of data [77]. The correlation studies allowed us to determine the relationship between the different features and the nature of the event. For instance, using the color code of the heatmap, we can observe the strong correlation between some email addresses and the size and the attached files in figure 9.7. We reuse these heatmaps in the feature selection steps.

This exploration of the dataset allows us to get some insight into it. Besides, we need to pass by a preprocessing

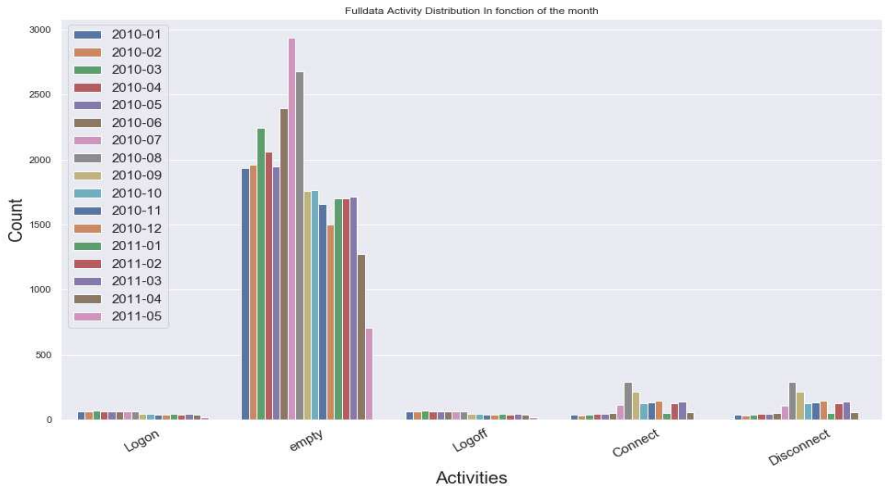

Figure 9.4: Activity distribution in a training data sample, scenario 2

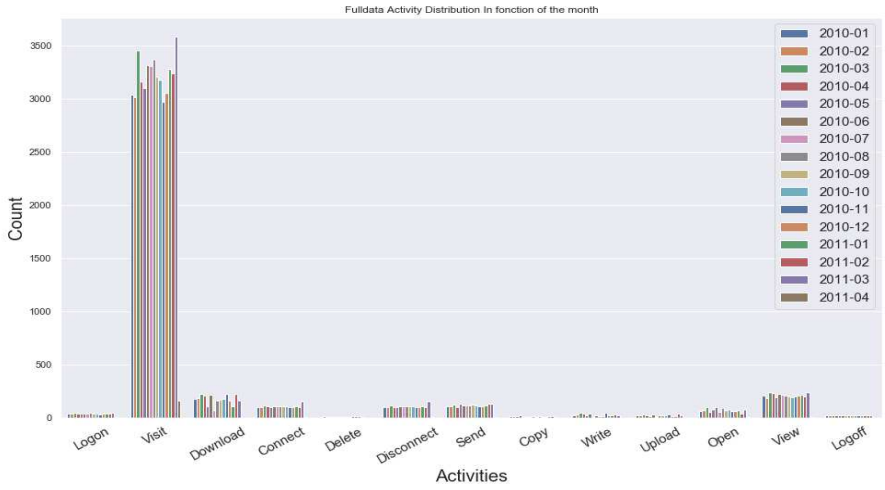

Figure 9.5: Activity distribution in function of the month for the R6.2, scenario 2

steps to prepare the data for our UCEL framework.

9.3 Prepossessing for the version R4.2

Specifically for the *R4.2* subset, after the data visualization, we begin a preprocessing phase in order to create a training, validation, and testing set. At the begging of our test, specifically with version *R4.2*, we didn't use specific feature selection techniques. We use the description of each scenario and the person correlation from figure 9.7 to determine the most accurate features. After doing preliminary tests, we consider that the number of features would not represent an issue for the classifiers.

Hence, we use the merged*R4.2* subset and perform classic numerical encoding to transform the categorical

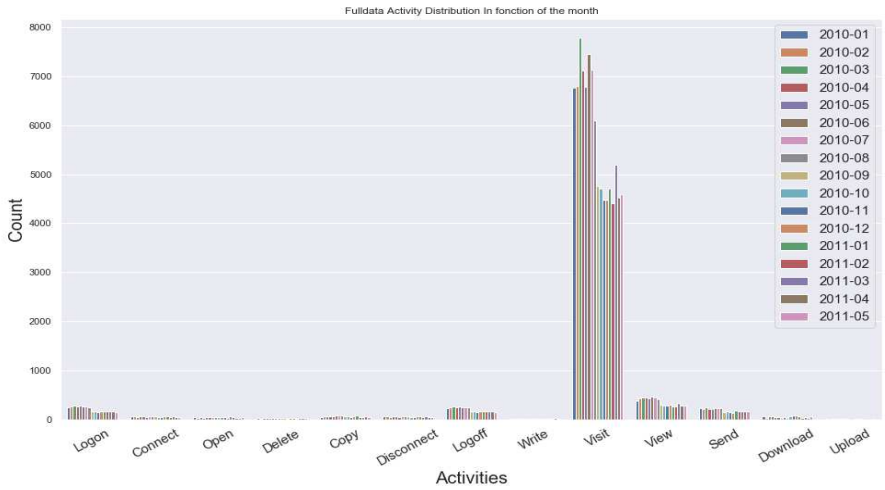

Figure 9.6: Activity distribution in function of the month for the R6.2, scenario 3

features into a numerical component. Particularly for this subset, we consider all the users that are malicious for the attack scenario. Since we a subset which is based on only one user, UCEL will build a role profile and not a simple profile. We also perform standard scaling and normalization on this data. This will help some classifiers such as the artificial neural network in their computations.

To obtain the training, validation, and testing set, we split the data using a classic random splitting function using these respective percentages 60%, 20%, and 20%. The figure 9.9 and the 9.10 present the activity distribution of the training validation/testing data. With this visualization, we could see the sharing of the activities but no particular sign of attacks.

9.4 NLP based preprocessing for the version R6.2

For the *(*R6.2) subsets, the notion of behavior is more detailed than the previous versions. As we discussed before, there is more variety to users activities and also more features to consider. Preliminary to the test for the *(*R6.2) data subset, the methods applied with preprocessing of the *(*R4.2) was not sufficient to get good performance. These trials showcase that the notion of the behavior is more complex to grasp in the latest version of the CERT data. Besides, contrary to the *R4.2* subset, the *R6.2* subsets contain only one user by attack scenario. This gives enough sample diversity for *R4.2* but not for *R6.2*.

The advantage of this dataset is that it contains more features information. Hence it offers possibilities to apply more analysis and selection steps. At the opposite of *R4.2*, the activity feature is used on all the data files (see tables 9.1 and 9.2). This adds significant information such as the view on the email data, visit on the website, the number of downloads, and upload in the dataset. We focused on a unique user base behavior profiler for scenario 2, and in scenario 3, on a role-based user profile.

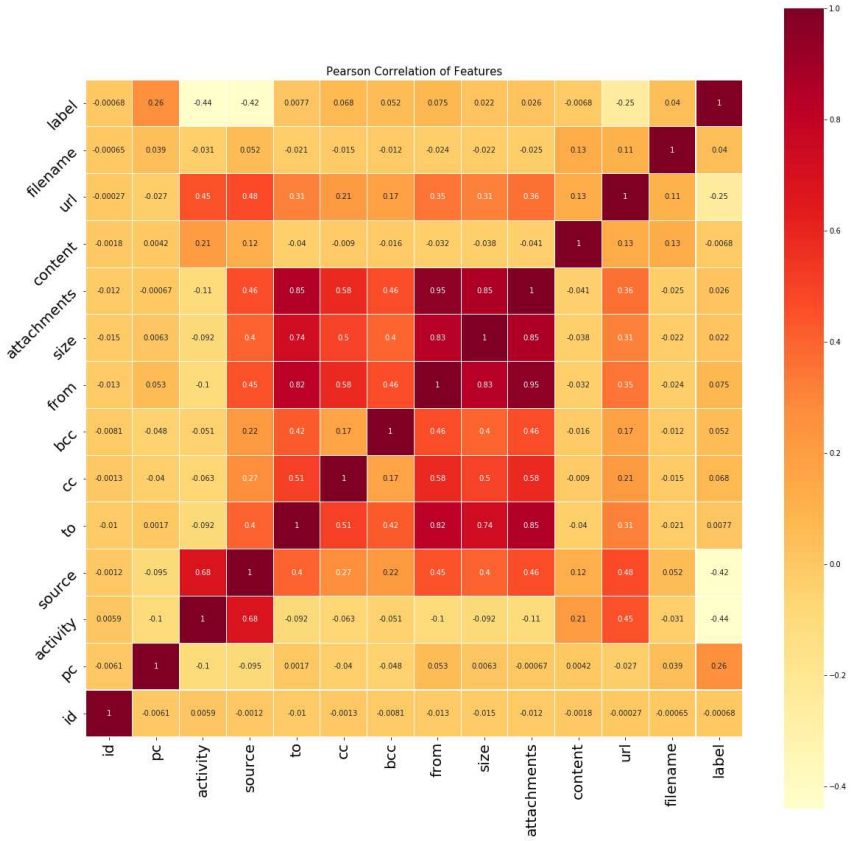

Figure 9.7: Pearson correlation for R4.2, scenario 2

The *email* and the *http* data have a content feature that contains extended strings. These represent the website and email textual content. For each data file with a content feature, we can use these strings full of information for our insider threat detection task. Hence, we proceed to natural language processing (NLP) on the email and web page content. These create new features that are some of the words used in these strings.

The goal of this NLP approach is to perform a lexicon or vocabulary analysis by checking if there is a divergence in the word frequencies during the period of normal activities and the attack scenarios. Hence, we used this analysis to create new features resenting the most uses word during the fraudulent email communication, website visit. More precisely, we choose the most common words in the *email*, *http* contents and object features, but also the URLs and file trees during an insider attack period.

This was especially important for attack scenarios 2 and 3 since they involved fraudulent email communication and the visit to a specific job website.

With the new feature and the event timestamp, we can visualize the dataset in the form of time series. Suppose

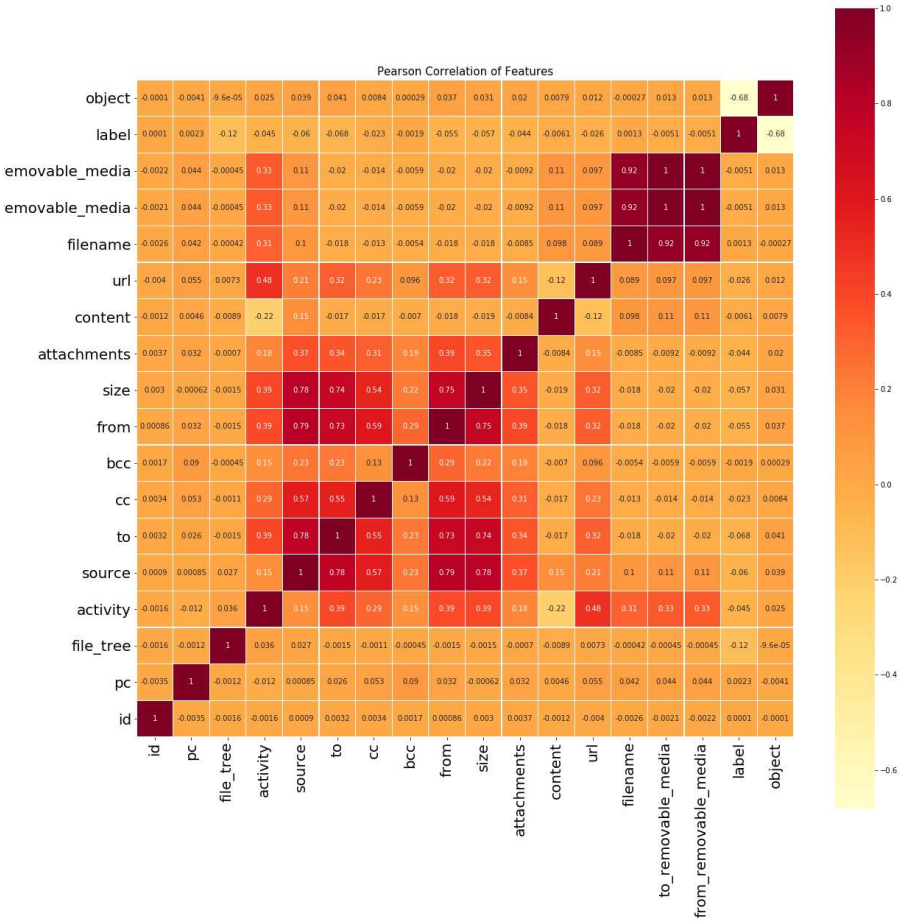

Figure 9.8: Pearson correlation for R6.2, scenario 3 before NLP

we focus on scenario 2, where the users begin to surf job websites, soliciting employment from a competitor, before leaving the company. In that case, we plot the frequency of the word 'project', 'jobsearch', 'clearance', and 'closing' during the period of data record obtained by applying the NLP. We showcase this visualization in the figure 9.11.

By analyzing the figure 9.11 we can see that during the period of the scenario 2 attack (between "2011-02" and "2011-03") the frequency of the chosen word increases dramatically. Hence these unusual usage frequencies can be indicative of this type of attack. This validates the choice of NLP preprocessing. It also gives us the means to exploit algorithms using time series analysis to deal with the insider threat problem. This type of algorithm could detect the strange behavior of some variables over time and flag anomalies using chosen thresholds.

We follow the same principles for the NLP-based preprocessing of scenario 3, which obviously create different features. We present in figure 9.12 the Pearson correlation of the new feature to the labels.

The result of the correlation study showcases the new feature correlations. However, we also remark that the number of dimensions of the problem drastically increases. This makes it hard to analyze the problem and might be

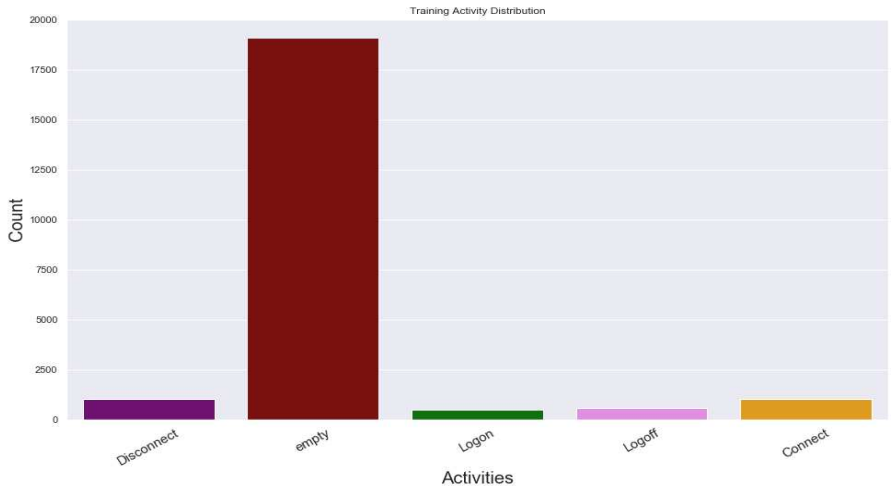

Figure 9.9: R4.2-2 Activity distribution in a training data samples, scenario 2

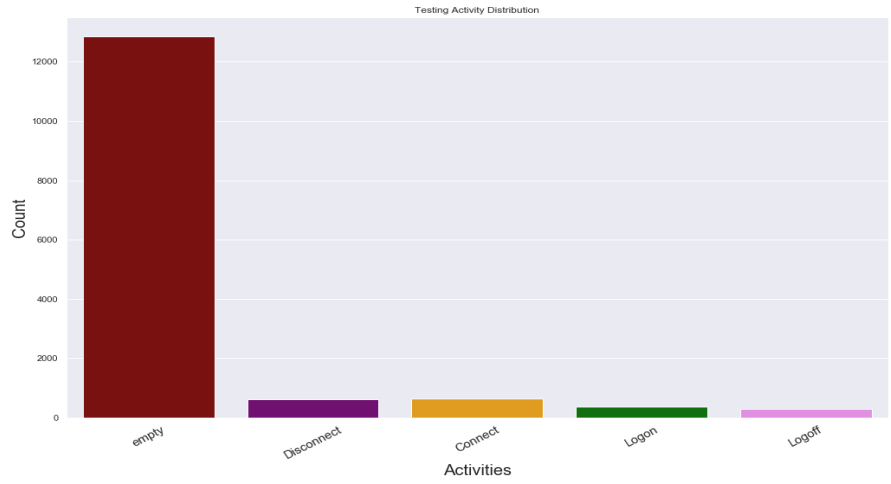

Figure 9.10: R4.2-2 Activity distribution in a validation/test data sample, scenario 2

an issue for the machine learning method (i.e., the curse of dimensionality). In order to deal with these issues, we applied some feature selection techniques.

9.4.1 Feature selection

Additionally, to the natural language processing, we also applied a classic numerical encoding, for instance to get the recurrence on the visit activities of URL or the send activities on to the same people. Using the NLP based preprocessing, we went from 14 to 70 features in the *R6.2* subsets. In order to avoid the curse of dimensionality and determine the best feature for the insider threat detection problem, we use different features selection techniques such as:

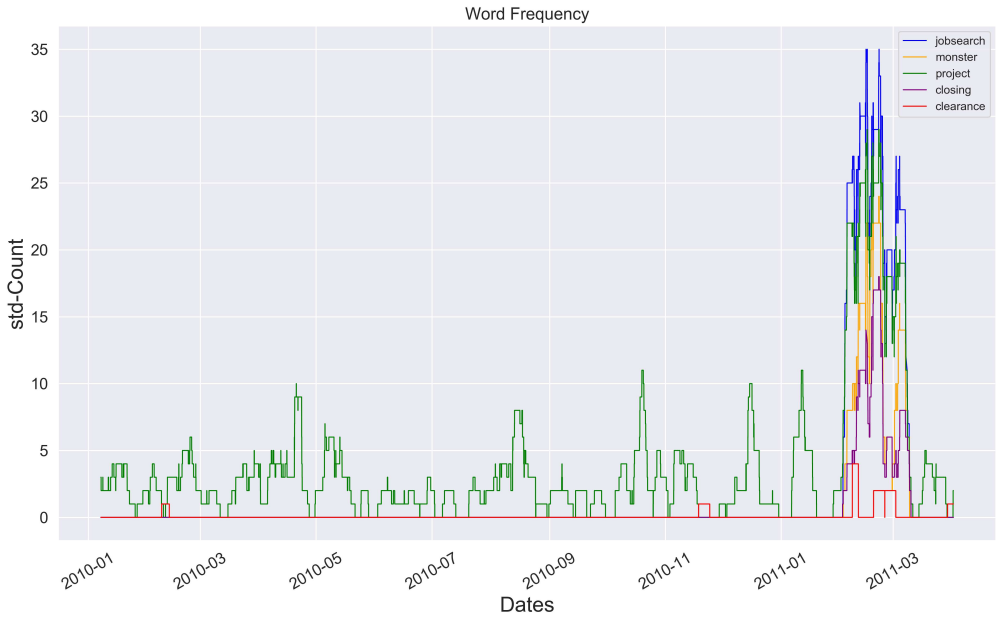

Figure 9.11: Suspicious word R6.2, scenario 2

- Pearson correlation [77]
- Chi-square [47]
- Recursive Feature Elimination [12]
- Embedded Lasso Ridge [78]
- Embedded Random forest [65]

We present in figure 9.13 the feature selected by every feature selection technique we applied. We used this technique for all the scenarios, but here we present only the result of scenario 2.

After all these steps, we then choose the most common features for a UCEL framework. Figure 9.14 present a boxplot of the most common feature selected by the feature selection techniques. We then applied a feature scaling and normalization operation to optimize our loss function's minimization.

We then create a train, validation, and testing set, however, using custom splitting methods inspired by the time series approach. This time the training set is made of the data activities before the first activity of the attack scenario. The test and the validation set are an equal split of the period between the first activities of the attack scenarios and the last recorded activity. This allowed us to simulate an insider detection system that have the habit of receiving normal activity before an attack occurs and that must make decisions when new strange activities occur. Hence, there are a normal activities in the training data and a mix of normal and abnormal in the validation and testing data. It is important to note the distribution between normal and abnormal activities is not uniform. For instance, in the case of scenario 2 the testing set represents approximately only 10% of the global dataset of the targeted user.

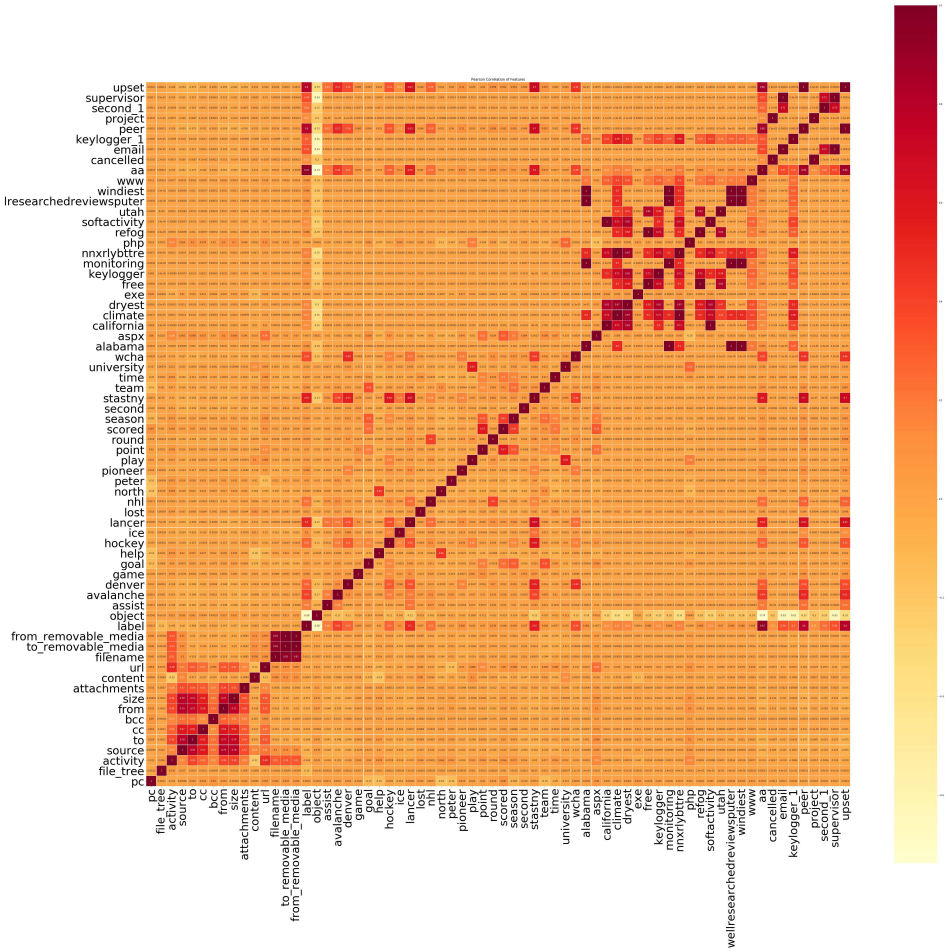

Figure 9.12: Pearson correlation for R6.2, scenario 3

In figures 9.15, 9.16, 9.17, and 9.18 we present a histogram of the different activities in a training data and the testing set respectively for the scenario 2 and 3.

	Pearson Corelation	Chi-square	Embeded_lr	RFE	Embeded_lgb
0	cancelled	avalanche	user	user	id
1	pioneer	hockey	pc	pc	user
2	dryest	ice	file_tree	file_tree	pc
3	bcc	lancer	activity	activity	file_tree
4	climate	stastny	content	content	activity
5	nnxrlybttre	wcha	url	url	source
6	ice	climate	object	object	to
7	second_1	dryest	climate	nhl	cc
8	avalanche	keylogger	nnxrlybttre	stastny	bcc
9	keylogger_1	nnxrlybttre	www	climate	from
10	hockey	softactivity	aa	dryest	size
11	supervisor	aa	email	nnxrlybttre	attachments
12	email	cancelled	keylogger_1	www	content
13	stastny	email	peer	aa	url
14	wcha	keylogger_1	second_1	email	filename
15	lancer	peer	supervisor	keylogger_1	to_removable_media
16	object	project	upset	peer	from_removable_media
17	peer	second_1		second_1	object
18	upset	supervisor		supervisor	assist
19	aa	upset		upset	avalanche

Figure 9.13: Feature selection table

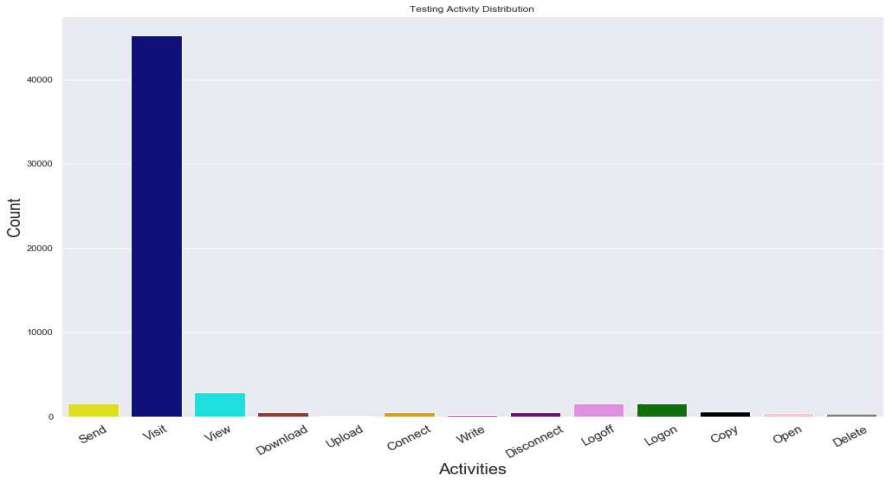

Figure 9.18: Activity distribution in the test and validation data R6.2, scenario 3

These figures showcase multiple divergences in the number of activities between the training, the testing, and the validation sets scenario period. This gave us information on the frequency of the activities in the different datasets. However, we do not find strange patterns from this visualization at the contrary of the time series representation.

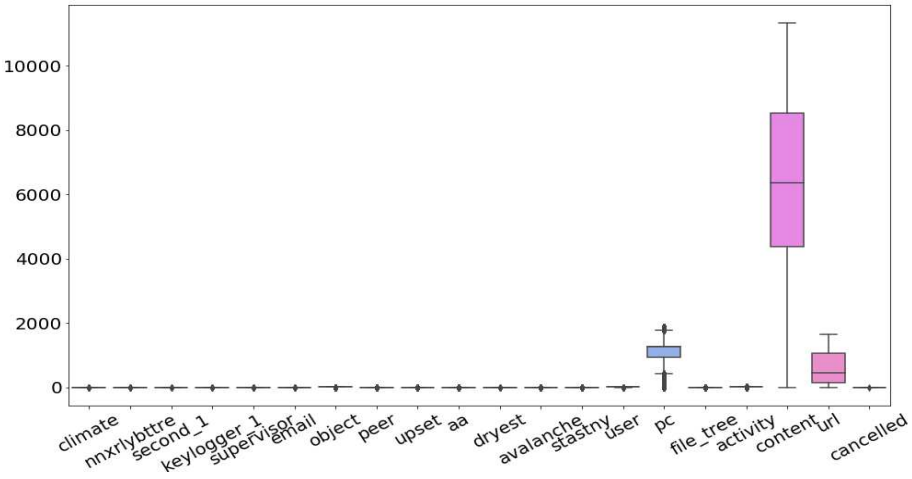

Figure 9.14: R6.2-3 Features boxplot after NLP

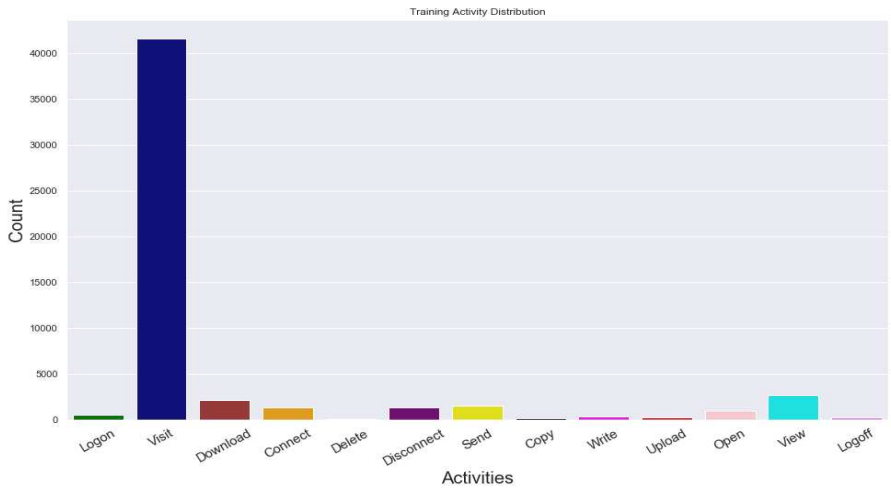

Figure 9.15: R6.2-2 Activity distribution in a training data sample for the Scenario 2

9.5 Activity frequency-based preprocessing

In the last section, specifically for the *R6.2* version of the CERT data, we remarked that the activity types vary in number according to the period of the year. We investigate, if this can be indicative of an attack. For that purpose, starting from the original user data, we buid a dataset with the daily activity occurrences . We have created a small algorithm that starts with the subsets of scenarios and calculates for each day the number of times an activity is performed.

These are the following activities:

- logon/logoff : User logs or leave to an endpoint
- Visit : User visit URL

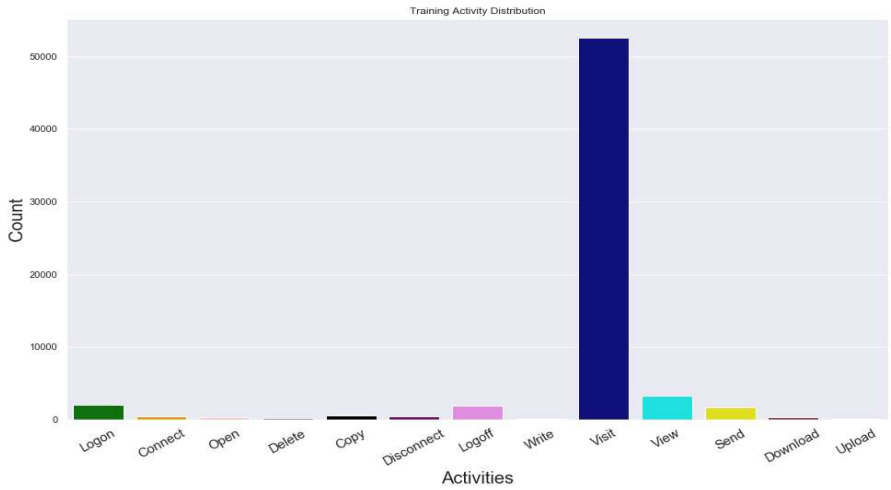

Figure 9.16: R6.2-3 Activity distribution in a training data sample for the Scenario 2

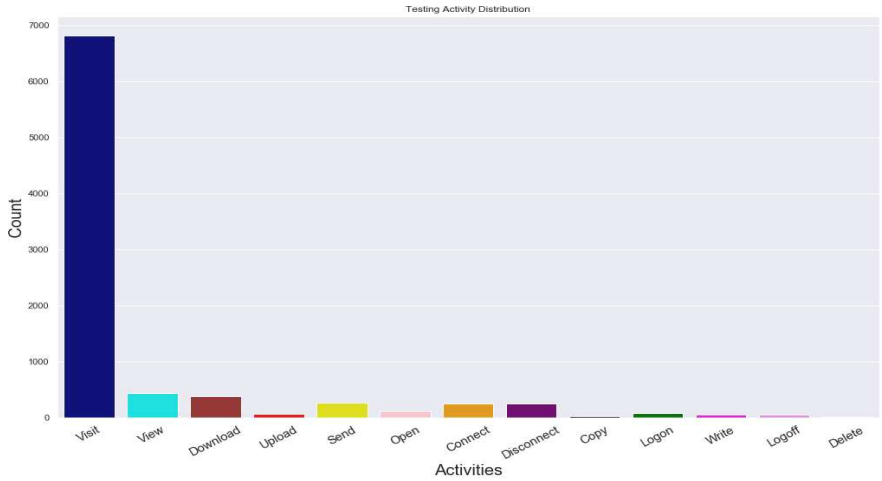

Figure 9.17: Activity distribution in the test and validation R6.2, scenario 2

- View: User view email
- Send: User send email
- Download/Upload: User upload or download file form an url
- Connect / Disconnect: User connect or disconnect a thumb drive
- Open/Write/copy/Delete: User Open, write, copy or Delete a file

We add the suffix '_num' to create the feature name. In this activity data frame, we have the number of different activities in a day. Hence, in a day, we computed the number of activities such as " logon", "logoff", "visit", "download." We then reapply the exploration process on the new data in order to get insights on the data.

The corresponding experiment is done for scenario 2. However, they can be applied to the other scenarios. It

is important to note that this reasoning can only be applied to user profiles, not on the role profiles. This approach counts the daily activities of users. In the role profiles, there are multiple users sharing the same roles. Hence, there can be an illogical variation of the users counts on vacation or particular activity patterns.

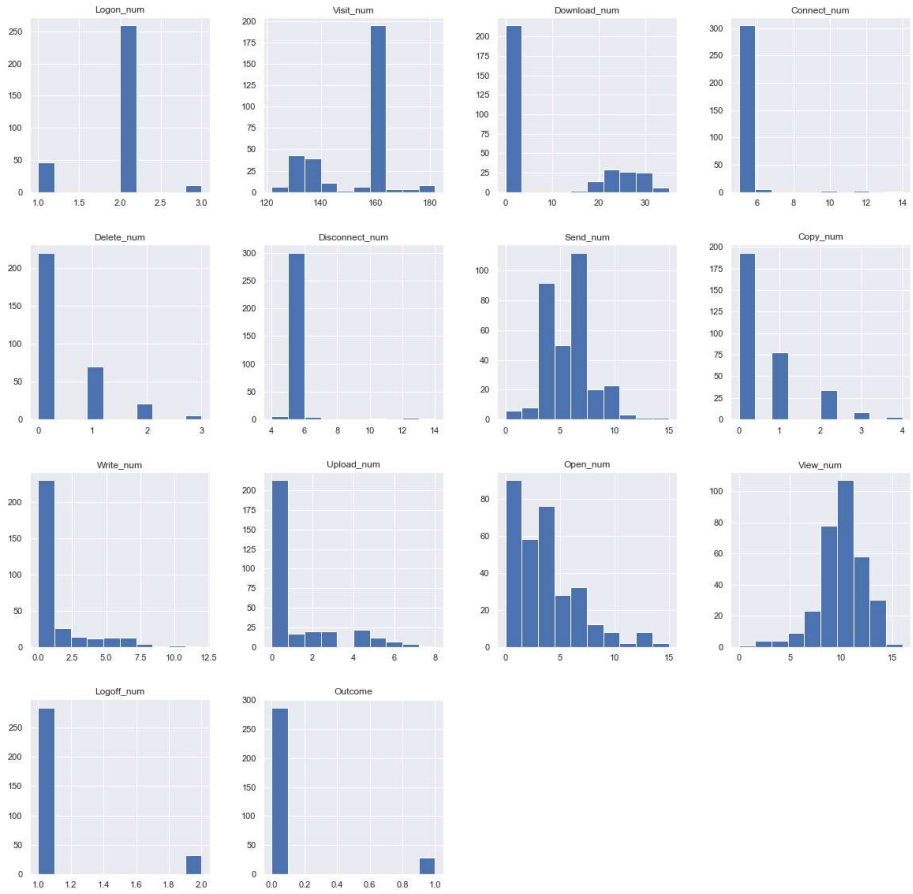

Figure 9.19: Activity data features histograms

We first do a univariate analysis of the data by plotting histograms of the data feature in figure 9.19. We then plot some feature in the form of the time series in the figures 9.20 and 9.21. If we look closely to the figure 9.19, 9.20, and 9.21, we can see in the distributions and the time series anomalous number of activities at the level of *Visit_num*, *Connect_num* and *Disconnect_num*. These anomalies are indicative of the malicious actions describe in scenario 2 (i.e., Visit Jobsite, and use thumb drive at markedly higher rates).

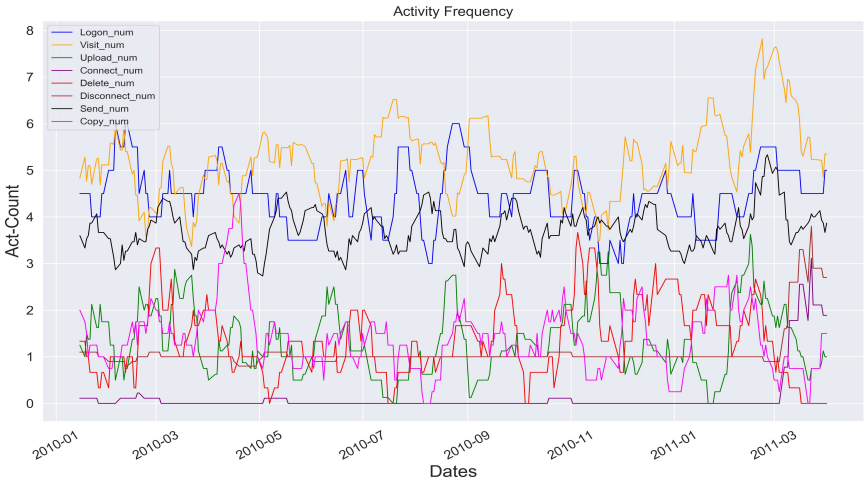

Figure 9.20: Activity data time series

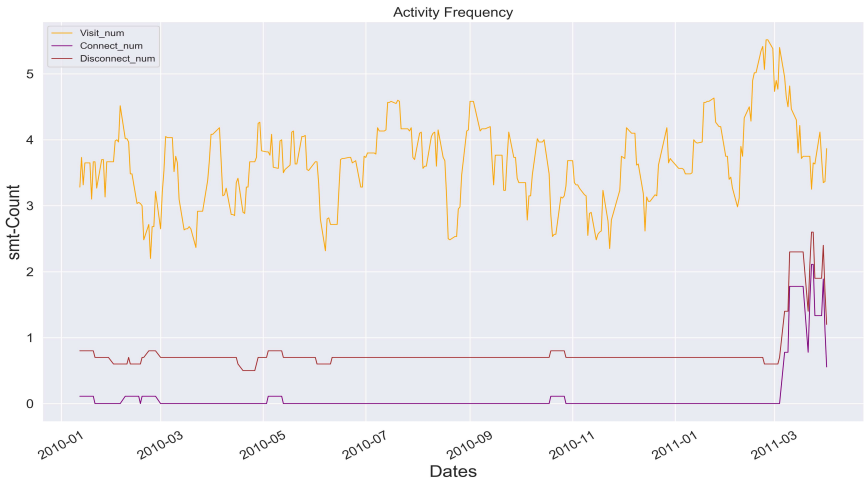

Figure 9.21: Specific Activity data time series

We finish by creating the boxplot of the feature of the new based figure 9.22. After the visualization, we apply the custom train, validation, and test distribution functions and apply the scaling and normalization process.

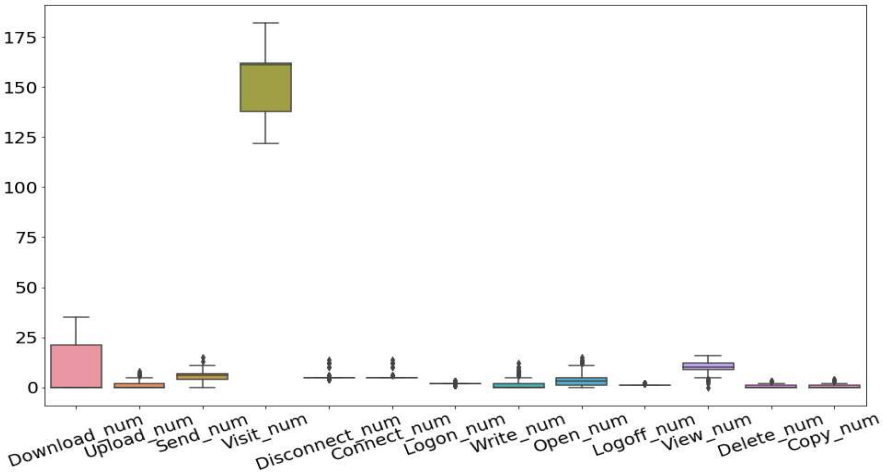

Figure 9.22: Activity data boxplot

9.6 Conclusion

This exploratory data analysis allowed us to analyze CERT insider threat data and to get insights for insider threat detection. We then use a preprocessing steps to build the training set needed to apply our UCEL approach. In addition, these steps gave insight into the type of methods we can use to detect insider attacks. We will also integrate these techniques to a global detection module in the following chapters.

Chapter 10

Classification experiments with UCEL on the CERT data

10.1 Introduction

To validate the proposed UCEL behavior profilers, we conducted firstly three experiments in a sequential execution context. To point out the benefits of the proposed approach, we used the *R4.2* (scenario 2). We also compared the framework with the classic boosting technique applied to the base methods. We then use the *R6.2* (scenarios 2 and 3) after introducing the PageRank and autoencoder based methods. Thereafter, we test the UCEL using the new activity occurrence dataset and data from other domains.

In the following section, we denote by $BP(L_1, \cdots, L_m)$ a UCEL-based behavior profiler composed of m learners $L_1, \cdots, L_m$, by $MultipleL(m)$ or $ML(m)$ a UCEL-based behavior profiler composed of m instances of the learner L and, by $MultipleL(i)$ or $ML(i)$ the i^{th} instance of $ML(m)$, for $i \in [1, m]$. We use here both UCEL-based behavior profiler and UCEL framework to designate a behavior profiler based on the UCEL approach. For the presented experiments the input values of the algorithms 5.6 are the following $l = 4, 6$, $q = 9, 20$ and $\theta = 1.0$.

10.2 UCEL with anomaly detection co-methods

Here, the objective is to highlight the interest of the UCEL approach with anomaly detection methods as co-methods. We start this study with the one-class support vector machine, the isolation forest, the robust covariance, and the local outlier factor.

OcSVM [66] is a variant of support vector machines (SVM), which is a large margin classifier that establishes a planar decision boundary between positive and negative examples. It can be considered as a way to apply SVM

to an outlier detection problem. The decision boundary of OcSVM opts for a spherical approach instead of planar approaches, as the support vector data description methods (SVDD) [36]. The goal is to find in a high dimension space, which is the minimal circumcising hypersphere that comprises only the good observations.

IForest [54] is a variant of decision trees and random forest algorithm. We discuss before that this method uses a comparison of the depth of the tree branches to spot anomalies. The shorter branches are indicative of anomalies.

The robust covariance/elliptic envelope (Robcov) [60] method uses the assumption that the normal data belongs to a known Gaussian distribution. The outliers are spotted when they are too distant from the center of the distribution.

The local outlier factor (LOF) [8] method studies the neighborhood of a data sample. It measures the local density of a given sample with respect to his neighbors. Outliers samples are detected when they present less density than their neighbors.

10.2.1 UCEL compared to boosting

Figure 10.1 presents the comparison of a Robcov(10) (10 instances of the robust covariance) and an individual boosted Robcov. As we stated before, the instances of Robcov are distinct in their hyperparameters choices. The individual boosted Robcov is the one with the best hyperparameters (i.e., the Robcov with the highest AUC when trained and tested alone). It is iteratively boosted with its own FP/FN. For this experiment, Robcov(10) outperforms the Robcov with the best hyperparameters only boosted.

The figure 10.2 presents the same type of comparison with a behavior profiler built with four different anomaly detection co-methods (4AD), and the four boosted base-methods working independently. The considered co-methods are Robcov, LOF, IForest, and OcSVM.

Despite a lightweight performance loss at the seventh iteration, UCEL provides an overall better AUC. This is probably due to an unlucky bag selection. The presented curves highlight that the UCEL-based behavior profiler presents a better max AUC than each individually boosted base method.

The behavior profiler uses the other co-methods insights to improve the starting performance of the global method to a greater extent. It leads to a more precise global learner, helps to reach a state of convergence faster, and a better max AUC.

10.2.2 UCEL and its co-methods evolution through cycles

As a second series of experimentations, we check the benefit of UCEL approach to manage the bias and variance trade-off.

The figures 10.1,10.2,10.3 and 10.4 depict the AUC as a function of the number of iterations of the UCEL framework. These figures represent the AUC evolution of the co-methods training. The figures 10.3 and 10.4

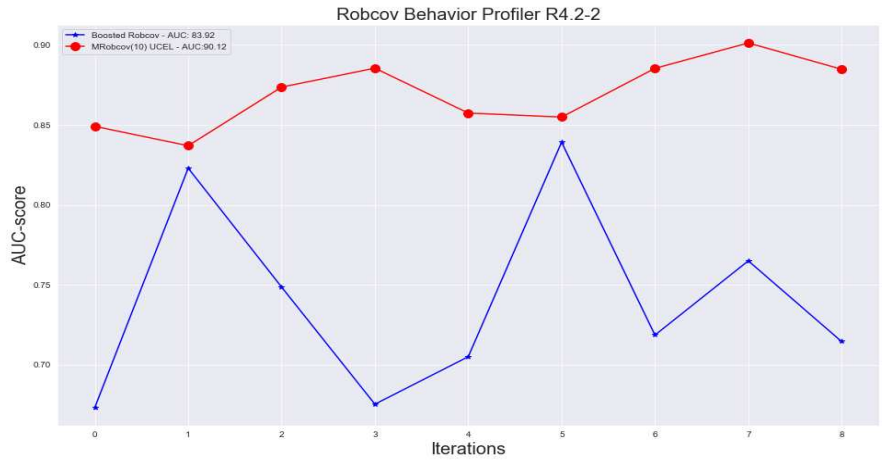

Figure 10.1: MultipleRobcov(10) vs the best individual boosted Robcov

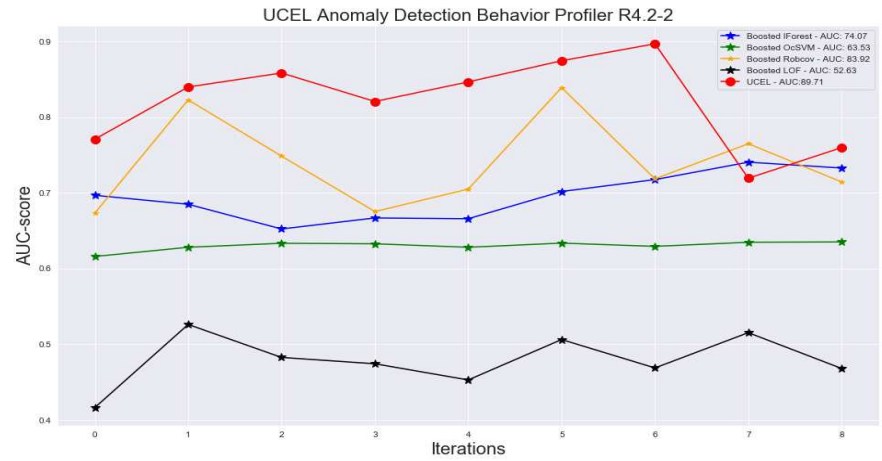

Figure 10.2: BP(4AD) vs individual boosted co-methods

represent respectively a UCEL execution of MultipleRobcov(10) classifier and UCEL(IForest, OcSVM, Robcov, LOF) with four different anomaly detection classifiers used as co-methods. For some co-methods, the score is pretty low after the first cycle. We can suspect that, individually, these co-methods suffer either from underfitting, overfitting, or poor calibration of the hyperparameters. UCEL boosts their initial low AUC through the iterations.

Here, we focus particularly on the peak accuracy of co-methods or in the WVC. In Adaboost [14], the weighted voting systems positively balance the classifier with low error rates and negatively the ones with high error rates. Here, we focus on the peak AUC of the learners. We will investigate in future works other weighting and restarting functions (e.g., additionally to the FP/FN, we will communicate the best hyperparameters).

Some co-methods showcase a little drop of performance after reaching their peak or oscillate between low and high values from an iteration to another. This is probably due to the choice of the hyperparameters. The OcSVM co-

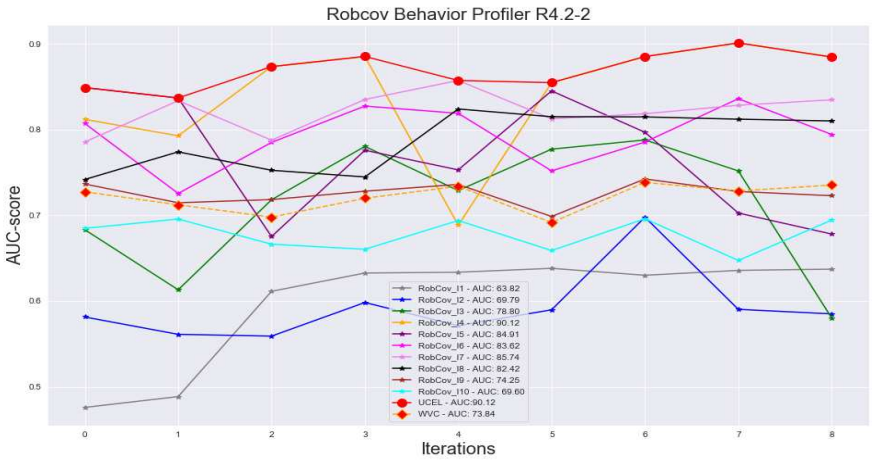

Figure 10.3: MultipleRobcov(10) and co-methods evolution through cycles

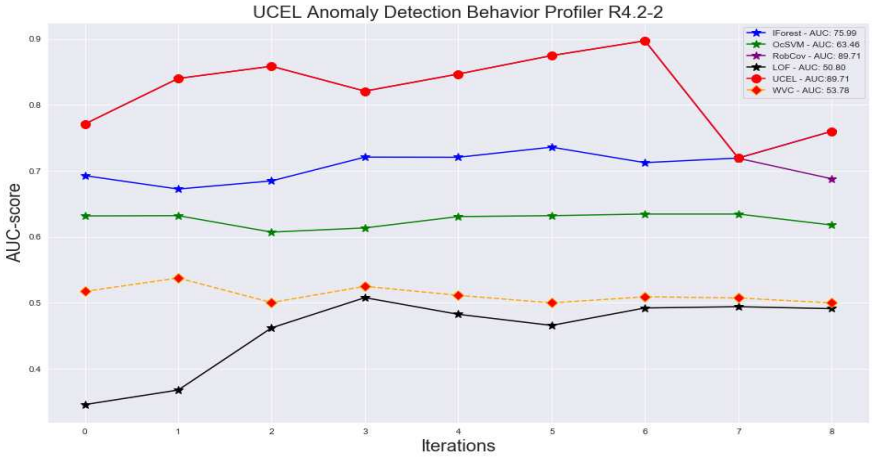

Figure 10.4: BP(IForest, OcSVM, Robcov, LOF) and co-methods evolution through cycles

method, in figure 10.4, does not seem to benefit from this boosting strategy after the second iteration. We believe that it is probably due to its tuning. However, we need to do further investigation, particularly at the level of its objective function and the establishment of the decision boundary when we inject new elements.

Haidar et al. [39] have proven that the efficiency of an injection of false-positive to OcSVM classifiers can improve its performance, however not in an iterative boosting manner. We also know that our model is sensitive to an unlucky random sampling. These cases can be spotted when the AUC is oscillating.

If the AUC threshold is not reached, UCEL selects the best method or *WVC* during the iterations. For this example, we choose the maximum number of iterations equal to $q = 9$. In figure 10.3 and 10.4 the precision threshold ($AUC = 0.99$) is never reached, but the AUC increases from 0.85 to 0.90 for 10.3 and, 0.77 to 0.90 for 10.4. If we focus on the figure 10.3, UCEL mostly improves the training accuracy for the co-methods with an initially low

AUC. This is a direct consequence of the use of this particular combination of boosting and bagging that improves weak learners training errors.

However, instances (9) and (10) of Robcov do not seem to be improving a lot by UCEL. This is indicative of a poor selection of hyperparameters and implies that tuning a classifier plays a non-negligible role in the performance of UCEL. Hence, except for instances (9) and (10), the co-methods starting with low training AUC get improved by the restarting process of injection of the wrongly classified element. This also indicates that even when two of the co-methods do not contribute to the classification performance, the UCEL approach still allows the other co-methods to get better classification results. However, it should be remembered that the objective of the UCEL framework is not the individual improvement of its co-methods but that one of the global methods.

10.3 UCEL with supervised learning co-methods

The goal here is to show the contribution of the UCEL approach with supervised learning methods as co-methods. The multilayer perceptron (MLP) [64] is a feedforward ANN used here for classification. It is very efficient to fit a non-linear function to a dataset. K-nearest neighbors (KNN) [32, 62] is a sample classification method that outputs class membership of a sample based on a popularity vote of the neighbor's samples. Quadratic discriminant analysis (QDA) [75] is a classifier using a quadratic decision boundary to separate the classes. The gaussian naive bayes (GNB) [45] is a supervised classifier based on the use of conditional probability and the Bayes theorem.

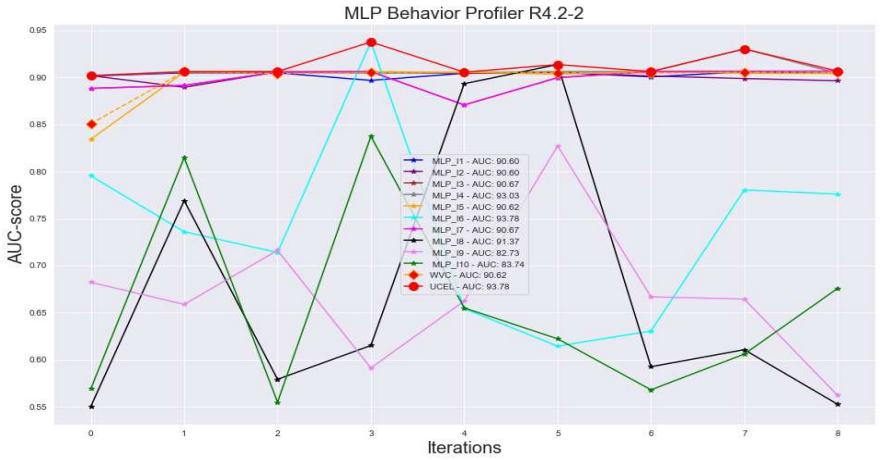

Figure 10.5: MultipleMLP(10) and co-methods evolution through cycles

The figures 10.5 and 10.6 respectively present 10 MLP instances and 5 different supervised classifiers. We recall that the goal of UCEL is to build a strong learner able to use the feedback from the composing methods. In this case, we notice that UCEL improves the co-methods with low starting AUC (e.g., instance (2), (6), (8) in figures 10.5) in figure and does not improve the ones with an already high score.

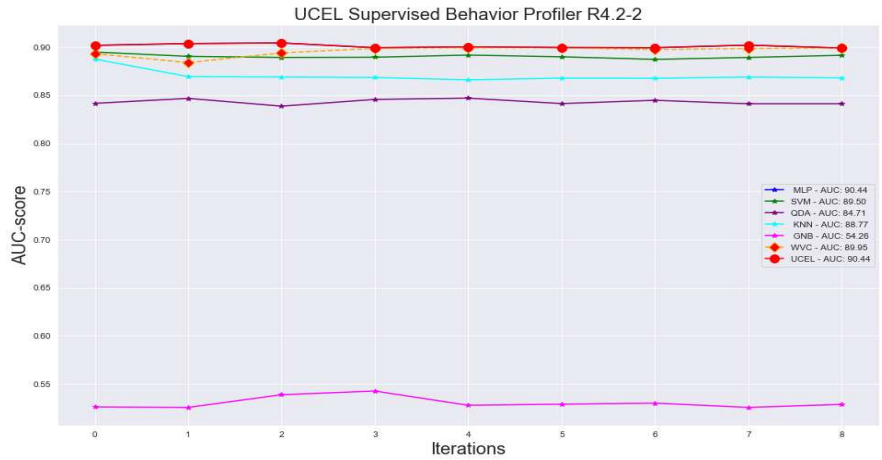

Figure 10.6: PB(MLP, SVM, QDA, KNN, GNB) and co-methods evolution through cycles

This is a consequence of boosting and bagging working well only with weak learners as base methods. Strong learners can get improvement using this strategy, but not to the same extent as weak learners[40].

In the figure 10.6, the KNN, MLP, and SVM classifiers are not improved by the UCEL process. Their AUC is rather good from the start and stable through the iterations. This is indicative of well-tuned classifiers for this problem. The GNB classifier is not improved at all by the process. As for the unsupervised methods, we suspect that this is due to bad hyperparameter tuning or poor feature selection for this method.

We also remark that the WVC of UCEL using all the co-methods produces the second-best results. Hence, the corresponding model was then chosen as the best model (i.e., MLP) of the behavior profiler. In the case of UCEL with starting high-performing classifier, the WVC provides almost always good results, but not necessarily the best result. The WVC ensures that the global learner profit from the experience of all the co-methods. However, when there are co-methods that perform very badly (i.e., GNB), it impacts negatively slightly the performance of the WVC, but it still provides an efficient ensemble learner.

10.4 Scenario results

In table 10.1 of the next subsection, we can observe that the best instance of MultipleMLP(10) is still suffering from overfitting since the train test score varies from $0.90 - 0.50$. The UCEL framework fixes this issue and helps to obtain a train-test score of $0.93 - 0.92$. In the example of figure 10.6 with five supervised learning co-methods, the AUC is 0.90, so pretty high for the individual co-methods. Despite that, UCEL adds an improvement of 1% to their scores. We can conclude that UCEL helps to improve the class prediction performance of the training and the testing error by managing the bias-variance trade-off.

Here, we can also see mostly satisfying results by examining the overall performance of UCEL-based behavior

profiler for the three considered scenarios of insider attacks.

Table 10.1 shows the result of the training and testing AUC without and with the UCEL framework. Let *4AD* represent an execution UCEL with four different anomaly detection methods and *5SM* a UCEL with five different supervised methods. The train-test (TT) results showcase high bias and high variance issues from the best method of the ten Robcov instances and the four anomaly detection classifiers without UCEL. This points out that UCEL helps to manage bias and variance trade-off to obtain better testing results.

Method	TT without UCEL	TT with UCEL
Robcov	0.61 - 0.53	0.90 - 0.89
4AD	0.75 - 0.52	0.89 - 0.89
MLP	0.90 - 0.50	0.93 - 0.92
5SM	0.90 - 0.90	0.90 - 0.90

Table 10.1: Train and test with and without UCEL

Table 10.2 presents the test results of *10Robcov*, *4AD*, *10MLP*, and *5SM* methods. In most of the scenarios, we tend to see better classification results for supervised methods than anomaly detection methods. This confirms the best strategy is to adopt the use of semi-supervised methods when the data is imbalanced and then use supervised methods when the companies dispose of enough feedback.

Scenario	10 Robcov	4AD	10MLP	5SM
1	0.95	0.95	0.95	0.96
2	0.90	0.89	0.96	0.99
3	0.82	0.64	0.98	0.93

Table 10.2: Test results for 3 types of insider attack

10.5 Graph-based insider threat detection using PageRank

We present here an insider threat detection method based on graph analysis and the PageRank algorithm. The next objective is to point out the interest of this method as a base learner for its integration in UCEL framework for anomaly detection. Graph analysis methods are another family of learning methods. They are useful for classification, clustering, and anomaly detection problems. Ultimately, our goal here is to use these methods as a co-method of UCEL approach. We make use of PageRank technique proposed by Yao et al.[79] as a means to detect anomalies in data.

Their goal is to use the PageRank scores of graph nodes as an indicator of anomalies. It is about starting with an activity dataset to build a graph by defining the vertices and the edges. Our goal is to create a user behavior model using this graph analysis method. We applied the same type of reasoning as a subgraph analysis and belief propagation to detect insiders. We then create a new classifier class. Since PageRank uses the statistical random walk principle, we can consider an employee's actions during a day as its walk.

Yao et al. [79] proposed to use a proximity graph built from the original datasets. A proximity graph is a graph where vertices are connected when they satisfy geometric properties. In this case, proximity refers to a spatial distance. There is an edge between two vertices when they are close enough (compared to a chosen threshold in the euclidean space). They propose in [79] to use three following types of graphs.

- KNN-Graph: A k nearest neighbour (kNN) graph is a graph $G = (V, E)$ with vertices $x_1, ..., x_n$ belong to $\mathbb{R}^d$. We define $dist_k(i)$ to be the distance from x_i to its k^{th} nearest neighbour. An edge (i, j) exists if and only if $dist(i, j) \leq dist_k(i)$.
- ϵ-Graph: An ϵ graph is a graph $G = (V, E)$ with vertices $x_1, ..., x_n \in \mathbb{R}^d$. An edge (i,j) exists if and only if $dist(i, j) \leq \epsilon$.
- EMST graph: a euclidean minimum spanning tree is a graph graph is a graph $G = (V, E)$ with vertices $x_1, ..., x_n \in \mathbb{R}^d$.The edges in G form a concentrated tree, while $\sum_{(i,j)\in E} dist(i, j)$ is minimized.

After building such a graph, each edge weight is calculated by considering the proximity of vertices in the graph. We built a transition matrix using these weights. With the graph volume, a custom teleport vector is also created. Then, the PageRank of the chosen proximity graph is computed.

The personalized PageRank is a version of the PageRank where the teleport vector is biased, making a surfer jumps to a chosen subset of pages when he decides to stop following the links. This vector ensures the uniqueness of the PageRank vector x.

Here the PageRank score is used as an indicator of an anomaly. The idea is to assign larges weights to the edges of close-by points. According to [79], the random walk transition probabilities are proportional to the values of the edge. The nodes (the data samples in our case) with close neighbors get higher chances to be visited. Hence the anomalies are the points with the lowest probability to be visited.

The most important part of this process is determining the hyperparameters (number of neighbors, damping factor, type of distance) of the anomaly detection methods. In their article, Yao et al. focus on the usage of the ϵ-graph. In this thesis, we will mostly focus on the KNN-graph, and we will apply grid search and other cross-validation techniques to determine the hyperparameters, contrary to the rules of thumbs proposed by the [79]. The choice of cross-validation will allow maximizing the training and testing score. In figure 10.7 we present a example of the graph representation and in figure 10.8 a general representation of the algorithm.

Let $x_1, \cdots, x_n$ be the matrix of observations, f be a weight function and x computed PageRank vector. The algorithm 8 allows anomaly detection by using KNN-graph and personalized PageRank (PPageRank). It is important to note that in this algorithm, it is possible to do grid search cross-validation to obtain the best value for α afterward. To use the PageRank as an indicator to detect anomalies, we add to use a threshold value, which is a sort of decision boundary between normal and abnormal PageRank. To choose this threshold, we plot the PageRank distribution of the KNN-graph (see figure 10.9).

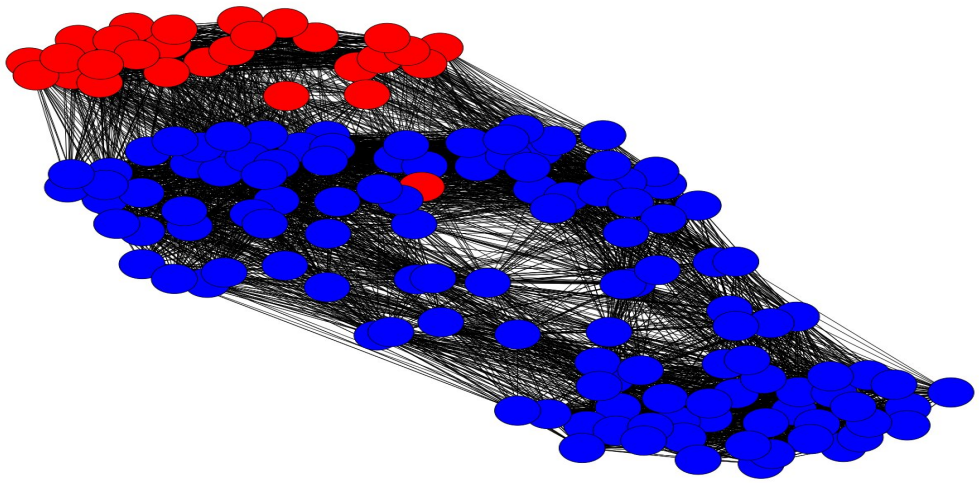

Figure 10.7: KNN-Graph sample

Algorithm 8 Anomaly Detection using KNN-graph and PPageRank
KNN-PP(in: $[x_1, \cdots, x_n]$, f; Out: x, Anomaly labels)

1: **Compute** pairwise distances among measurements
2: **Determine** vicinity criteria to form a proximity graph
3: **Do** A grid search cross-validation function to determine the optimal hyperparameters K
4: **Build** The KNN-graph to obtain the adjacency or the distance matrix
5: **Apply** the weight function to turn the distance matrix to obtain the similarity matrix W
6: **Build** The Teleport vector using the distance matrix
7: **Normalize** W to get transition matrix P
8: **Pick** α as the approximately know contamination parameters
9: **Compute** stationary state $Ax = x$, with $A = \alpha P + (1 - \alpha)G$, to obtain x.
10: **Plot** The pagerank distribtion
11: **Sort** in ascending order and output the top few points as anomalies or pick detection threshold using a plotted distribution

We test this algorithm and obtain an AUC of 0.83, with $k = n/2$ and $\alpha <= 0.97$. We choose α by checking the percentage of the regular activities in the dataset. However, this method's results are struggling if we use the full data.

We also test a variant of this approach using the *Scikit-network* library from the work of Bonald et al. [5], and the *scikit-learn* library. We started by building a version of the above algorithm. However, this approach is semi-supervised. Hence, it needs a number of labels to give good results. For a 50000 sample dataset, we provide 3000 labels and $k = 51$ to obtain an AUC for 0.85.

The AUC results of the Bonald approach are showcase in the figure 10.10.

In the case of using bagging with this method as the base method, we get an AUC of 0.88. This is indicative of the possible positive impact of applying UCEL, as it combines bagging and boosting forces.

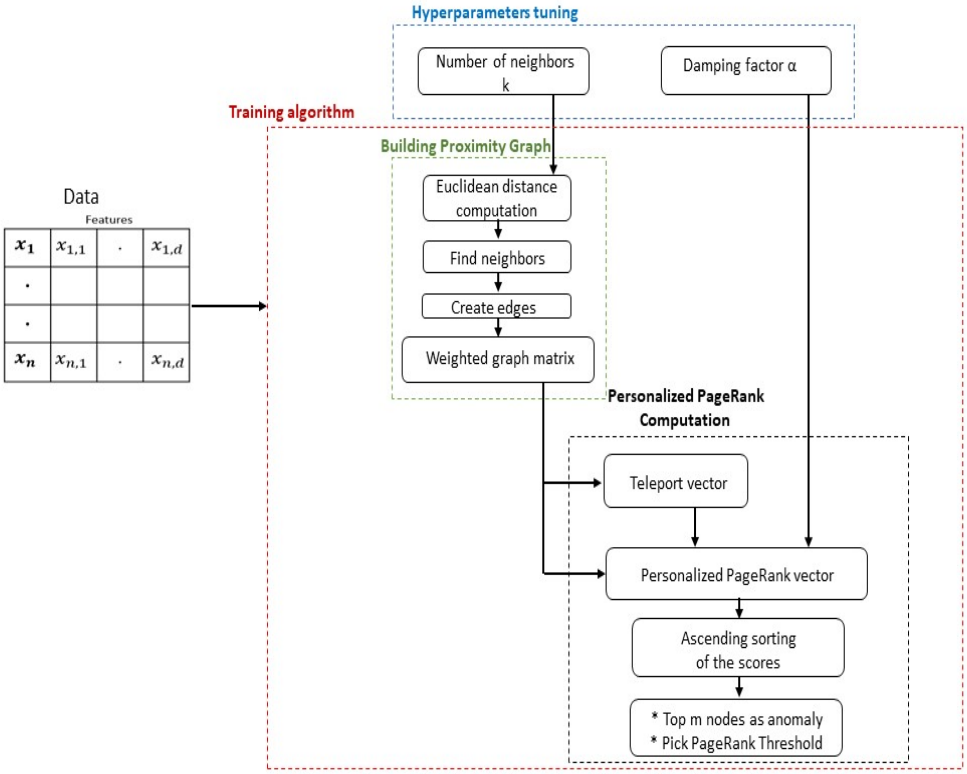

Figure 10.8: PageRank generalized approach

10.5.1 PageRank based method class and UCEL test

We build a classifier class following the same architecture of the *scicit-learn* library. We hence have a *fit* and a *predict* function. The fit methods only store the training data. The *predict* method works more like a *fit_transform* method since it calculates the adjacency graph using KNN and computes the PageRank, and classifies the data using the threshold. The number of neighbors, the threshold, the damping factor, the tolerance, and the number of iterations are the hyperparameters of this method.

For the experiments presented in figure 10.15, we used the CERT dataset (*R6.2*) (scenario 2).

The figure 10.11 depicts the evolution of the AUC over the cycles of a multiple PageRank with four instances. We can observe that the initial AUC of the co-methods is all slightly improved (approximately from 0.90 to 0.95) by the UCEL framework even though it oscillates through the iterations. The AUC of the PageRank co-methods is initially high ($AUC \geq 0.90$). Hence the improvement is small compared to the previous UCEL performances. The oscillations with peaks appearing on the PR curves are typical of eigenvalue calculation methods (PageRank here) when these are poorly located. But, as in the previous experiments, these oscillations can also come from the random sampling of the bags.

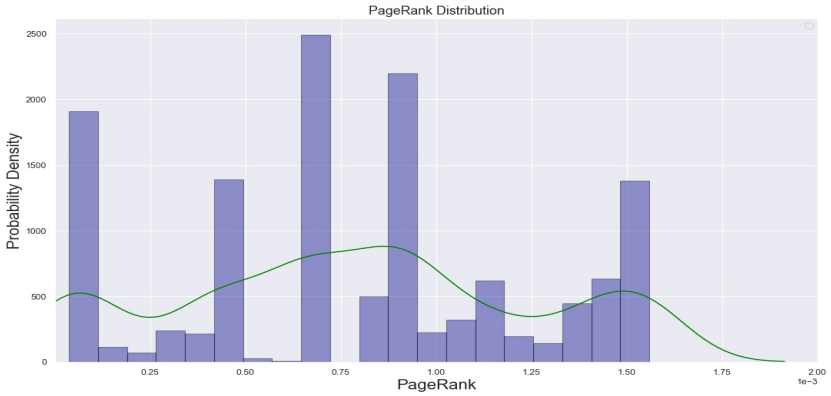

Figure 10.9: PageRank distribution

10.6 Autoencoder and LSTM based insider threat detection method

The objective here is to show how we can use an unsupervised autoencoder with LSTM cells to detect insider attacks. We also analyze its contribution to the UCEL framework.

Recurrent neural networks (RNN) are a type of artificial neural network that is good to learn temporal relationships [34]. The terms recurrent are associated with this method because it can retain the hidden state's information and propagate it over time. Some application examples of RNN are time series analysis, sentence completion, stock price prediction, language translation, self-driving car navigation systems, and music generation. In RNNs, concepts such as forward propagation and backpropagation are made through time. This method passes through multiple states to build a model.

$$y(t) = \Phi({x_{(t)}}^T.W_x + {y_{(t-1)}}^T.W_y + b) \tag{10.1}$$

In the equation 10.1, Φ is an activation function, $x(t)$ is an input, W is a weight matrix, b is the bias term, and $y(t)$ the output. This equation represents the output of the RNN neuron for a single instance. LSTM (Least short-term memory) is a type of RNN using gating cells with a combination of activation functions (fully connected layers) such as *sigmoid*(σ) and *tanh*, multiple gates (entry point in a neural network layer) for the model states.

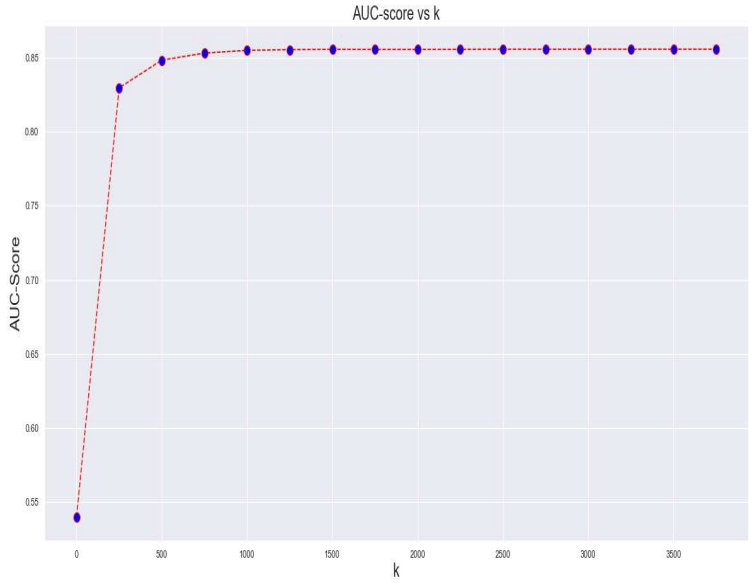

Figure 10.10: K-Validation

$$
\begin{aligned}
i_{(t)} &= \sigma({W_{xi}}^T.x_{(t)} + {W_{hi}}^T.h_{(t-1)} + b_i) \\
f_{(t)} &= \sigma({W_{xf}}^T.x_{(t)} + {W_{hf}}^T.h_{(t-1)} + b_f) \\
o_{(t)} &= \sigma({W_{xo}}^T.x_{(t)} + {W_{ho}}^T.h_{(t-1)} + b_o) \\
g_{(t)} &= tanh({W_{xg}}^T.x_{(t)} + {W_{hg}}^T.h_{(t-1)} + b_g) \\
c_{(t)} &= f_{(t)} \otimes c_{(t-1)} + i_{(t)} \otimes g_{(t)} \\
y_{(t)} &= h_{(t)} = o_{(t)} \otimes tanh(c_{(t)})
\end{aligned}
\tag{10.2}
$$

In the equation 10.2, we show the computation of the long term stated $h_{(t)}$, the short term state $c_{(t)}$ and the output $y_{(t)}$ of an LSTM basic cell. $h_{(t)}$ is also know as the hidden state. it is obtained in function of previous states and the input data($f(h_{(t-1)}, x_{(t)})$). $i_{(t)}$, $f_{(t)}$ and $o_{(t)}$ represent respectively the functions that control the input, the forget, and the output gates. $g_{(t)}$ is the main function that deal with the $x_{(t)}$ and $h_{(t)}$([34]).

This combination of gates is able to retain the basic pattern of the data and forget unessential parts. We can use these principles to retain important characteristics in user activity data. In a gating cell, there is a multiple module charge to retain or to forget information. We can use multiple instances of RNN as base methods for our UCEL approach. The use of this method is inspired in part by the unsupervised approach proposed by [76] although it is possible to use it in a supervised manner.

Here, we test a combination of RNN with LSTM cells and an autoencoder. Autoencoders are artificial neural

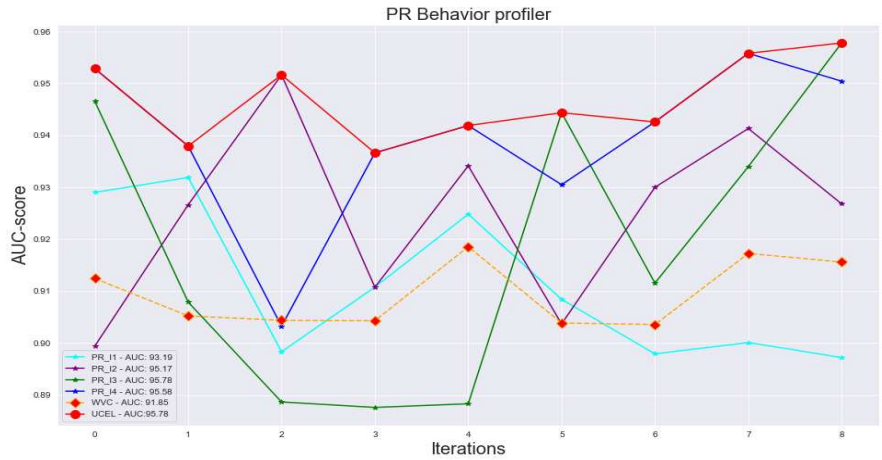

Figure 10.11: MultiplePR(4)

networks that learn and encoded a representation of the data. Our goal is to use the indication given by the autoencoder's reconstruction error to detect anomalies. This approach works following the assumption that regular activity reconstruction error will be different from abnormal movements. Hence, we build an unsupervised autoencoder with LSTM cells to detect insiders in the form of a new unsupervised classification class. The figure 10.12 showcases the type of autoencoder model we implemented using the *(*TensorFlow) library. In the last chapter, we used exploratory data analysis on the CERT dataset. We found that applying a time series analysis on some features allows detecting insider attacks for (scenario 2). This solution based on a recurrent neural network is built to detect unusual feature variations linked to time. With this method, we obtained an AUC of 0.85.

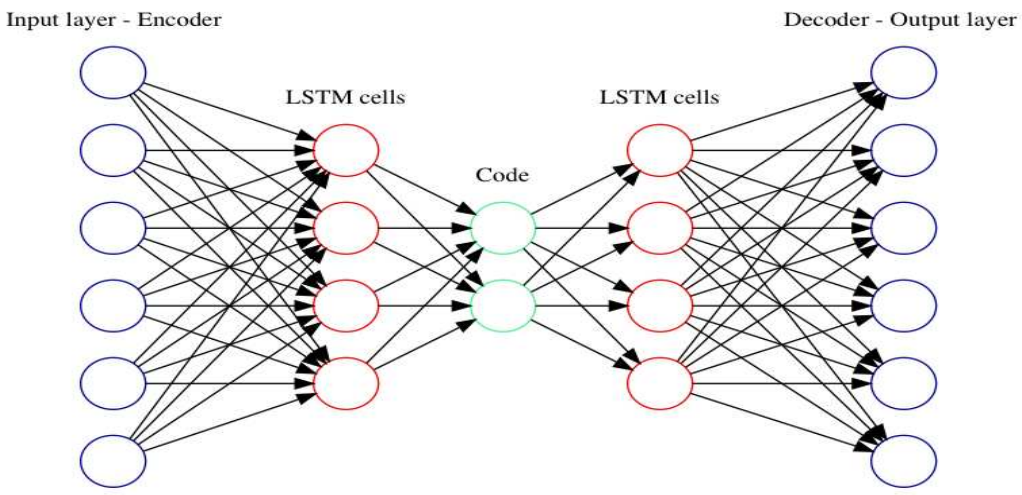

Figure 10.12: Autoencoder LSTM Model

10.6.1 Autoencoder LSTM class and test

To reach this score, we create a new classifier class with *fit* and *predict* function. The *fit* function starts by training the method using the activity data. Next, we can check the reconstruction error distribution to choose the less probable error as the detection threshold (see figure 10.13).The *predict* function compares the reconstruction error of the test data to the threshold to classify the data. The threshold, the number of iterations, the number of layers and nodes are hyperparameters of the methods.

Figure (10.14) presents a representation of the chosen threshold and the reconstruction error of all the data. Again, using only bagging for this method, we can improve the AUC to 0.90. This also proves the possible positive impact of UCEL with this base method.

The figure 10.15 depicts the use of LSTM as co-method in UCEL context. We can notice the contribution of this co-method to the improvement of the UCEL framework.

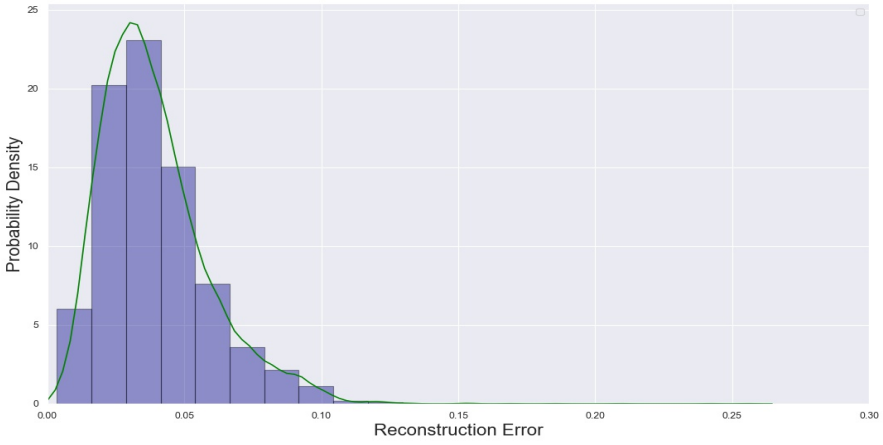

Figure 10.13: Reconstruction error distribution

10.7 UCEL with new PageRank and Autoencoder methods

Here, we present the first test of UCEL with the PageRank based method (PRBM) and the autoencoder LSTM based method. We used the version *R6.2* (scenario 2) for this experiment.

The figure 10.15 represents AUC evolution of UCEL with six different co-methods OcSVM, IFores, Robcov, LOF, PR and LSTM (see LSTM in the next section). We can observe that the initial AUC of co-methods, except IForest, Ocsvm are improved by the UCEL framework. The improvement on PR and LSTM is small (approximatively from 0.83 to 0.95) because they have a high AUC at the beginning. It should be noted that even an originally performing well co-method like PR is improved by UCEL. We notice here that the improvement of the PR co-method coincides

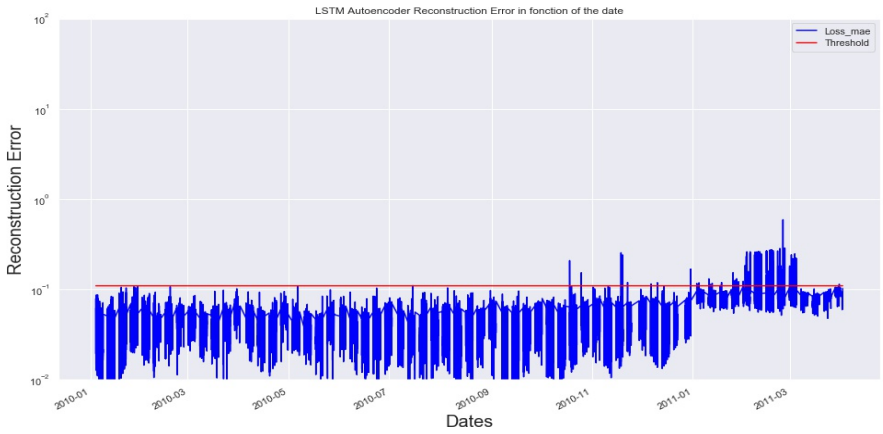

Figure 10.14: Reconstruction error in function of the activity date

with its selection as the best of the co-methods during the iterations. Robcov and LOF are also improved however their AUC is oscillating. The rest of the co-methods are not improved by the process.

Figure 10.15: PB(4AD, PR, LSTM)

10.8 Activity dataset test

In the figures 10.16 and 10.17 we present the results of the activity dataset.

For (scenario 2), in the figure 10.16, we found the best method is the PR through the iterations. All the other methods are improved by UCEL but not to the same extent. The top AUC is at 0.94. It is good accuracy; however, we obtain an AUC of 0.96 with the NLP Based preprocessing.

For (scenario 3), in the figure 10.17 at the first iteration, the PageRank and the OCSVM have an AUC of 1.

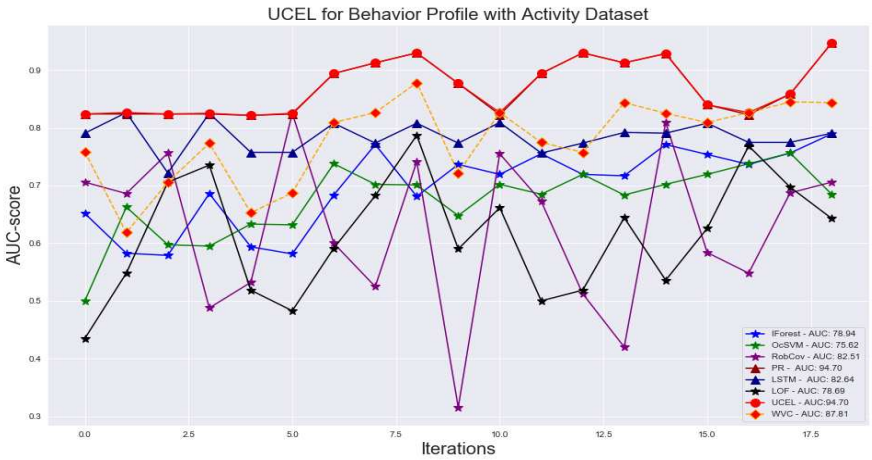

Figure 10.16: UCEL applied to the activity dataset, scenario 2

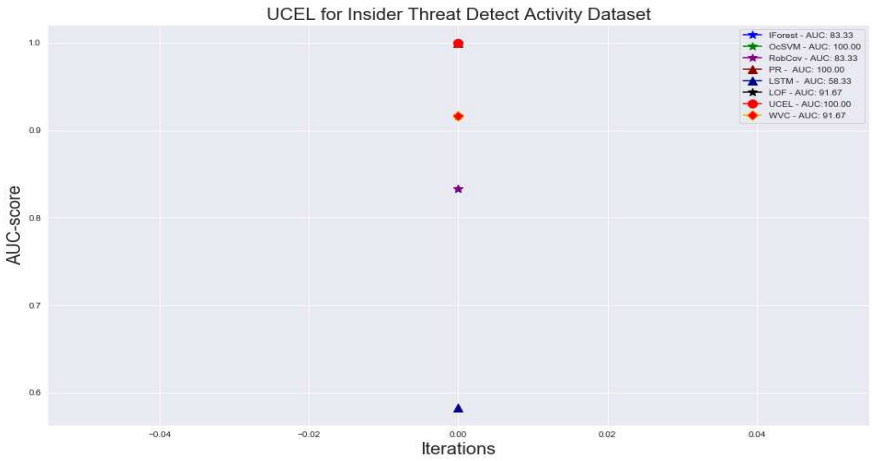

Figure 10.17: UCEL applied to the activity dataset, scenario 3

Hence, the iterative process doesn't start. On some occasions, the iterative process is not necessary. Therefore the best method is chosen at the first iteration. The activity-based preprocessing allows us to outperform the previous result of the UCEL framework for this scenario.

10.9 Oversampling techniques comparison

We also compare the oversampling technique use by UCEL in figure 10.18. This comparison showcase that the synthetic minority oversampling technique (SMOTE) gives better performance than the random oversampling (RO) techniques. However, using UCEL with SMOTE takes more time since it deals with bigger training sets than the RO techniques.

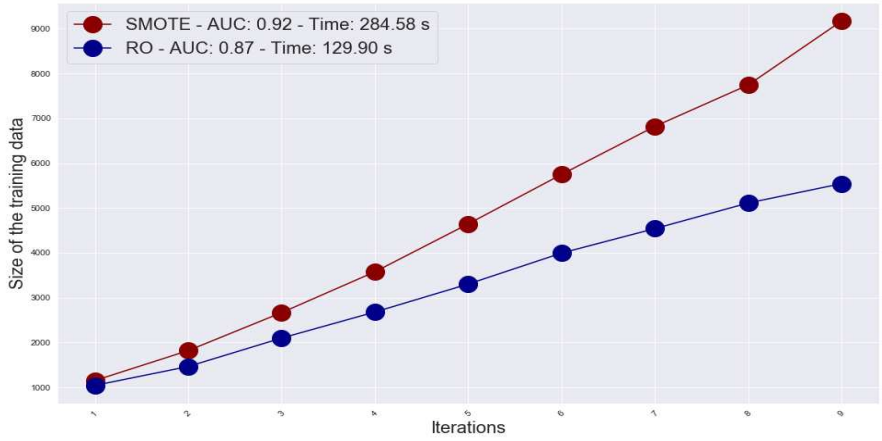

Figure 10.18: Oversampling techniques comparison

In both cases, the size of training size increase in function of the number of iteration. This leverages again the importance of high performance computing systems to improve the computation performance of UCEL.

10.10 UCEL applied to general classification problems

In this section, we present some experiments we did with UCEL applied to other domains. These tests help to showcase the usefulness of the UCEL framework for classification problems in general.

10.10.1 Credit card fraud

As we presented before, in some other IT domains such as social networks, telecommunication, sale sites, search engines, and recommendation systems, fraudulent user activities are an issue that cybersecurity experts are continuously facing. Similar in some aspects to the insider threat problem, this issue leverages the use of behavior analysis. In these fields, the term user behavior modeling refers to the study of users behaviors to classify them as usual or fraudulent. These actions are behavior anomalies, same as insider activities. With some considerations, the fraudsters can be seen as the equivalent of the insiders of these domains.

We discused in 3 before the fraud detection and the insider threat detection have some common point. They can mostly be translate to classification problem with a high data imbalances between the normal class and the abnormal class. This makes it a good candidate for UCEL based on anomaly detection co-methods.

In this case, UCEL is used to detect fraudulent credit card operations. In the used dataset, there is 0.172% of fraudulent transaction [18, 19]. We can see in figure 10.19 that UCEL improves the AUC of all the co-methods, except for the IForest method. This is probably due to poor optimization of the hyperparameters. In this case, we

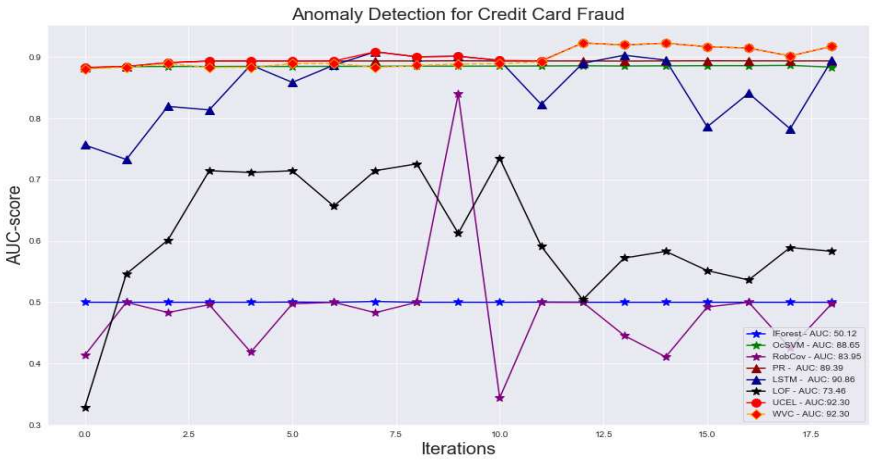

Figure 10.19: UCEL for credit card fraud detection

also observe the same AUC oscillations that are seen in the insider threat detection profiler.

10.10.2 Seismic bumps

In this case, we try to use UCEL on a dataset out of the domain of cybersecurity. Since credit card fraud can be attached to a cyber fraud problem, we can argue the domain share common attributes to the insider threat.

For this experiment, we use data for seismic bumps prediction [71] from the UCI repository. This dataset is used to forecast high-energy seismic bumps into a coal mine. This forecasting problem can be seen as a classification problem using the seismic measurement before a seismic bump occurs.

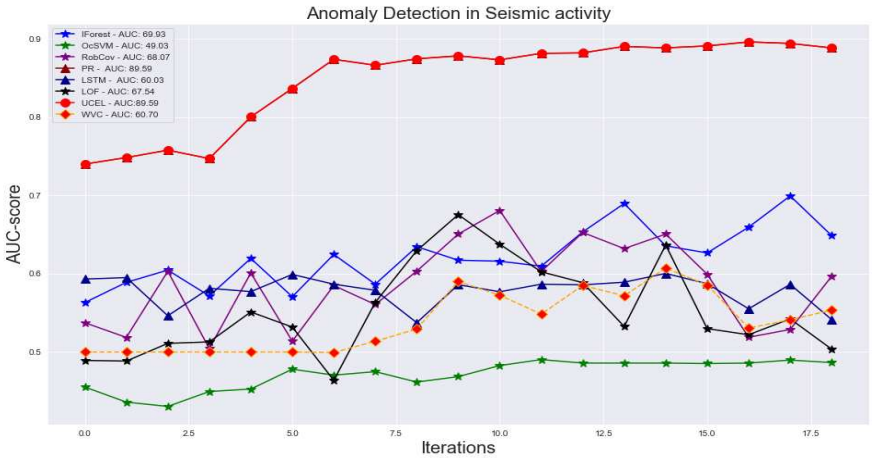

Figure 10.20: UCEL for seismic bumps classification

In this case, figure 10.20 we observe that UCEL improves the initial performance of the co-methods. However,

the PageRank method outperforms all the co-methods. We can make the argument that the PageRank algorithm is more suited to deal with these issues. To test that theory, we launched an instance of UCEL using only the PageRank method.

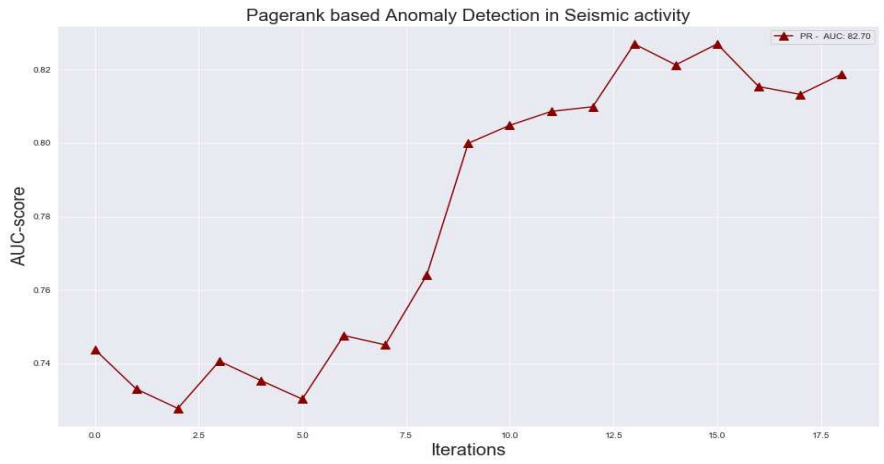

Figure 10.21: UCEL Based on PR for seismic bumps classification

The result of this experiment figure 10.21 proves that the PageRank-based methods can profit from UCEL working alone. However, even if they perform at a lesser level, the other co-methods positively impact the AUC of the PageRank method. The AUC improves from 0.82 to 0.89 using the collaboration mechanisms of UCEL.

10.10.3 Titanic passenger classification

In this example, we use the well-known *Titanic* dataset [26] that helps to deduce the deceased passenger in the function of their features such as their *passenger-class*, *sex*, and *cabin-number*. In this case, the classes (survived or deceased) are mostly balanced. Hence, we use UCEL with classic classification methods such as MLP, SVM, QDA, and KNN. We also add the semi-supervised label spreading method(LBS) based on a label propagation algorithm using the affinity matrix of the normalized graph laplacian. However, here we are still dealing with binary classification problems.

In this case, we also see an improvement due to the UCEL framework. We can discuss that the original feature selection and optimization of hyperparameters can be better to get better initial accuracy for this dataset. However, we decide to make these tests with a few previous optimizations to see the contribution of UCEL in extreme and different cases. Overall, UCEL showcases a positive impact on all the co-methods, with the WVC, choose as the best method figure 10.22.

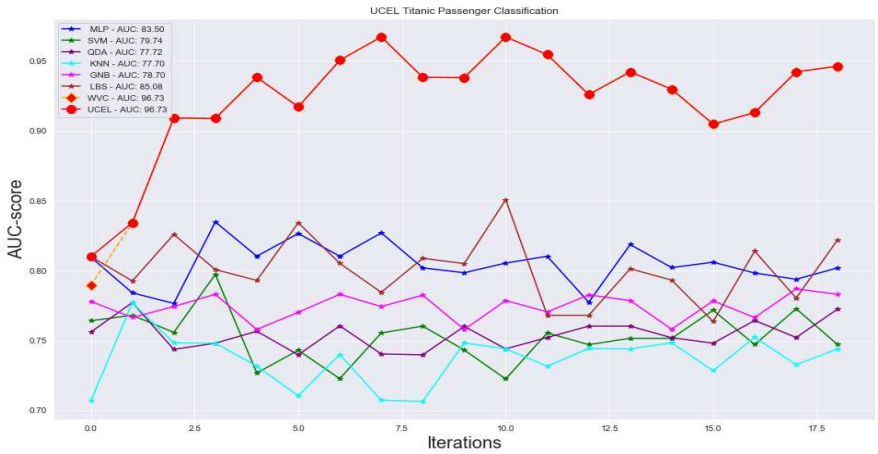

Figure 10.22: Titanic passenger classification

10.10.4 Fetal cardiotography classification

With the following two examples, we investigate the impact of UCEL on a multiclass classification problem. For that purpose, we use data from the life science domain. This dataset is made of 2126 fetal cardiotocography (CTGs) that were automatically processed. The diagnostic features were measured. Obstetricians analyzed these measurements. They reach a consensus on the classification label and the diagnostics [2].

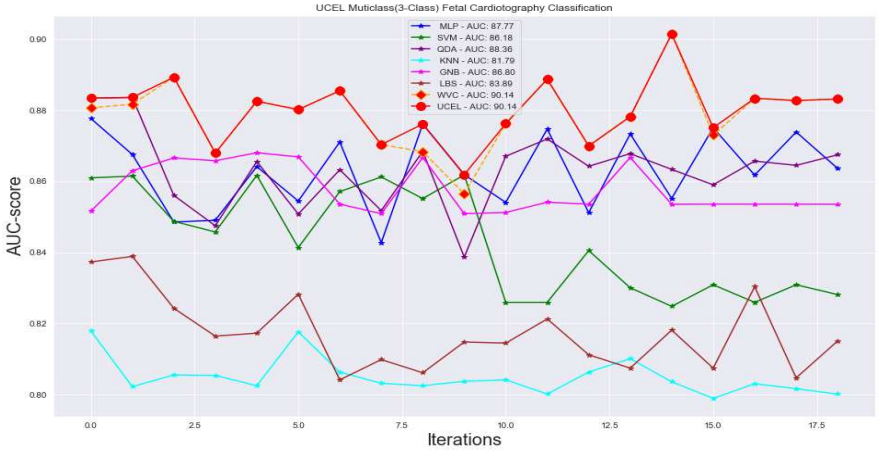

Figure 10.23: UCEL for fetal cardiotography

In this case, figure 10.23 we have a three-class classification problem. We can observe the same improvement of the classification performance as the binary classification issues since the iterative processes improve the initial AUC. This proves that this framework has a positive impact on classification problems in general, and it is not limited to binary classification and anomaly detection problems. We obtained an AUC of 0.90 for this experiment.

10.10.5 Car state evaluation

For the last dataset, we use a car evaluation dataset [69]. This is also a multiclass problem with four classes that characterize the state of the car. This dataset has the role of helping a potential buyer. The four different states are unacceptable, acceptable, good, or very good.

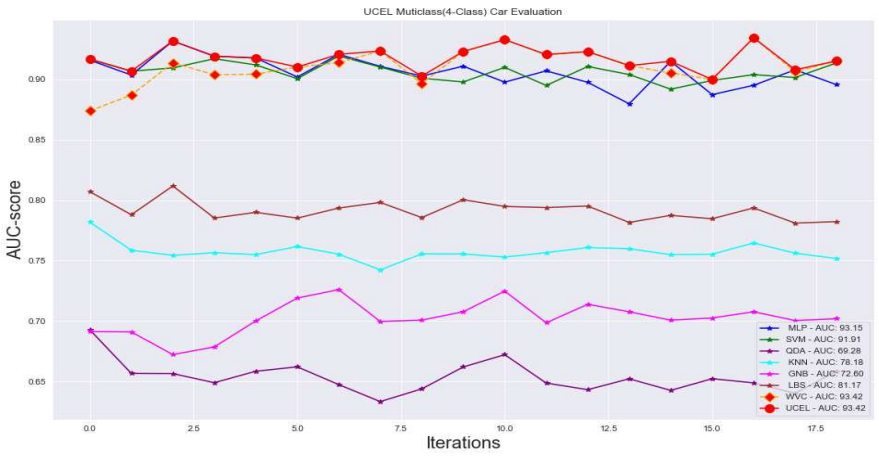

Figure 10.24: UCEL for car evaluation

For this experimentation, figure 10.24 UCEL also showcases good results. We obtained an AUC of 0.93. We also remark that the MLP, SVM co-method are the best methods in this case. Here we need further investigation to determine the cause linked to the underperforming classifier.

10.10.6 Accuracy comparison with original works

As a means of comparison, we present the AUC obtain by UCEL against the highest accuracies obtained by the studied dataset in their original papers. The results are listed in table 10.3.

Dataset	Original Paper	UCEL
Credit card fraud	0.86 AUC	0.92 AUC
Seismic bumps	0.971 Acc	0.8959 AUC
Titanic	0.862 Acc	0.96 AUC
Fetal cardiotography	(100% sensitivity, 99% specificities)	0.90 AUC
Car evaluation	70.3 Acc	0.93 AUC

Table 10.3: Accuracy comparison UCEL vs original work

10.11 Alarm system test

In this section, we present the test of the risk scoring algorithm of the overall insider threat module. This test was done with the *R6.2* scenario 3. We chose to use the role-based behavior profilers since it is a more general test than the user behavior profiler and will help to validate the overall behavior modules. We present the risk scoring result in figure 10.25. We specifically used two risk scoring methods based on the analysis of the daily user activity. We used the supervised and unsupervised behavior profiler, with the NLP and activity occurrence-based preprocessing.

Best Method Scorer (BMS) This is based on the prediction of the behavior profiler with better performance. The AUC of the behavior profilers are compared, and the most precise of them is chosen to determine the nature of the new unlabeled data.

Advanced Weighted Scorer (AWS) This method is based on the pondered prediction of the classifier. Here we use the AUC of the methods during the training and validation steps. In order to weigh the prediction of the corresponding classifier. The mean of the pondered prediction is then computed to establish the risk score.

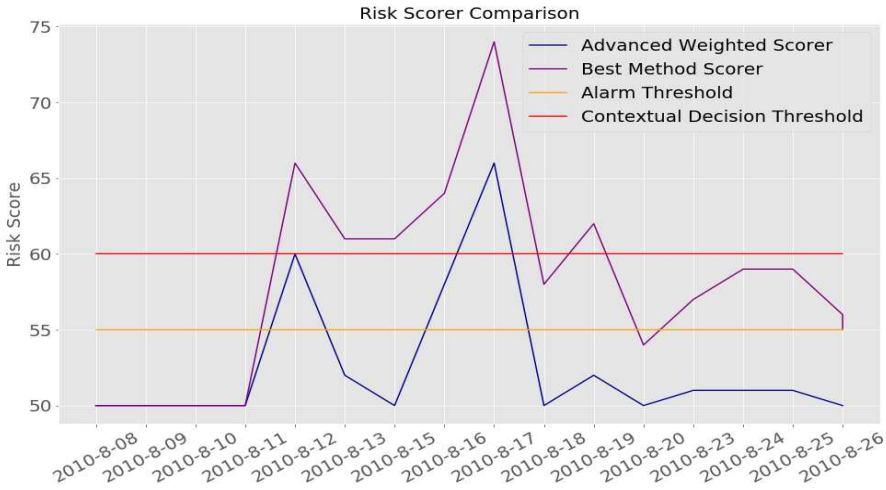

Figure 10.25: Risk scorer comparison

The days with anomalous activities are in the period between the *2011-8-11* and *2011-08-17*. We use the data before this period for the training and the validation of the behavior profiler. After this period, corresponding data is known as normal. If we analyze figure 10.25, we can observe both the scoring option detect the attack, especially on the *2010-8-17*. This date is the day with the most anomalous activities. However, we can observe that the risk score value of the BMS is higher than the result of the AMS. This means that if we pick a threshold to launch an alarm or take a contextual decision, the BMS is more susceptible to false alarms. Besides, in the period after the *2011-08-17*, the activities are normal, but the BMS gives a high score.

This proves that using an AMS type of alarm system has more advantages. For now, this system is based on real-time day analysis. The alarm systems receive the new event of the day and determine if the risk score of the monitored user. However, if they are not enough fraudulent activities on the day but spread through the wees, this system might miss the attack. Hence it is also essential to build alarm systems on more considerable periods such as week or month.

10.12 Conclusion

In this section, we test and validate the UCEL approach using mainly the CERT insider threat dataset. Overall, UCEL presented a good performance for all the proposed preprocessing approaches. However, the NLP preprocessing gave the best performance.

The goal of UCEL was to improve initial training results through the iterative process. This was verified by almost all the experiments. We tested this approach with multiple types of co-methods, multiple datasets, and different types of problems. The PageRank-based anomaly detection method and the autoencoder LSTM based methods perform the best (in this order).

It is also essential to note that we saw the most improvement for anomaly detection methods. Hence, we have the most improvement on the case of a highly imbalanced dataset. For the supervised methods, even though the performance improvement were not important, the WVC mostly help to use the ensemble learning voting principles to have better performances. In rare cases, the UCEL approaches stop at the first iterations. In those cases, the WVC also ensures to get the contribution of all co-methods.

Chapter 11

Parallel performance analysis

11.1 Introduction

In this chapter, we discuss the implementation of our contribution using high-performance computing methods. In 6 we presented multiple parallel programming models. At the beginning of our parallel test, we started with our co-methods (IForest, Robcov, LOF, Ocsvm). Originally we based the UCEL approach uniquely on the four classic anomaly detection co-methods. We studied and tested the speedup of the behavior profiler based on the algorithm 7 and compared it to a traditional client and server algorithm 6. We then switched from a UCEL framework with four co-method to six co-methods with the PageRank and the autoencoder-based classifiers. To investigate the performance bottleneck of our system, we also studied the average execution time of the co-methods.

We then test our approach using the asynchronous communication programming model. We also propose specific performance studies and the Multi-UCEL cases based on the PageRank and the autoencoder methods since they are the most performant methods and the most time-consuming. We finished by measuring the parallel implementation of UCEL deals with numerous user activity data training and validation.

11.2 Conditions of parallel experimentation

11.2.1 Metrics and hardware

In this chapter, we use different metrics to measure the performances of the parallel UCEL algorithms. We mainly test the execution times, the speedups, and the scalability of the different parallel programming models.

The execution time allows comparing the training time of the behavior profile implemented with a chosen parallel programming model. This helps to determine which of the programming model allows to have the faster ready insider threat, detection models.

The speedup is the ratio between the sequential execution and the parallel execution times. It allows quantifying the gain of speed when we use a parallel programming model. This is another means to compare the parallel programming models but from a different angle.

Finally, the scalability measure allows the analysis of the impact of the available computation resources on the performance of the programming models.

All the experiments were done on the Lille cluster of GRID5000, using a combination of node, CPU cores, and GPUs. GPU are graphical processing unit that uses their data-parallel properties to allows computing more efficiently data extensive problem. GPUs are commonly used for machine learning problems using neural networks, which is our case with the auto-encoder detection approach 11.8.

11.2.2 Libraries

In the previous chapters, we highlighted that the UCEL approach gives reliable results to detect insiders. To study the performance of the parallel version of the behavior profiler, we start by running implementations of the parallel algorithm of UCEL on the G5K platform.

We used the *Python* language and the *Multiprocessing*, *Multithreading* libraries to represent a client-server implementation 6. We use these two libraries to affect the training of the methods to the available core in the node.

For the algorithms 7, representing a UCEL model with synchronous communication, we used the *mpi4py* library to affect the co-method to the available nodes. The asynchronous model we present in the section 11.6 using also mpi4py function to manage the asynchronous communications.

11.3 Parallel data management

Since the CERT Data version *R6.2* was extensive, the *pandas* library was not able to withstand the data volumes of the event files. This is linked to the big data properties of the insider threat problem. At this point, our capacity to study and manipulate activity data was limited. We then try to find tools that can handle the lecture and the manipulation of a vast amount of data. We found and used the *Dask* library at the place of the *Pandas* libraries. The *Dask* library uses high performance computing techniques to read substantial data volumes in parts. It allows the extraction of information or a description of the data using computation functions. Since we based the solution of our problem on the use of individual employee or role activities, using *Dask* allowed us to read only the file related to the user we desire to study. It does not load all the base, hence avoid to fill the available memory.

11.4 Synchronous control node model approach vs client-server algorithm

11.4.1 First parallel models comparison

For this experiment, we used 9 nodes with 4 on the Lille cluster of G5K. We use the four classic co-methods with two instances each.

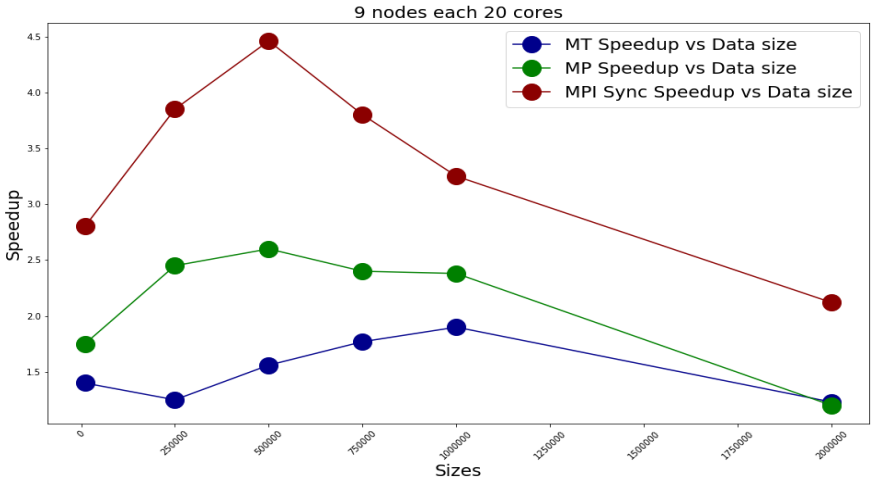

Figure 11.1: Parallel *behavior profilers* on GRID'5K

Figure 11.1 shows a significant speedup of 4.45. This speedup is reached using the algorithm 7 with the *mpi4py*. This can be explained by the fact that the co-methods are working in parallel for their training and testing phase. This confirms that *mpi4py* is more fit to benefit from cluster hardware than the other libraries. Even though the execution is always faster for the parallel implementation, we can notice a limitation on the speedup gains when the number of data records increases past 500000 events. However, after 500000 events, the performance loss is continuous for the *mpi4py* and *multitprocessing* parallelization libraries. We observe the same phenomenon with the multithreading libraries after 1000000 events.

We believe that this drop in performance is due to the multiplications of the communications and the synchronization steps. The communication of data becomes costly for the algorithm. The co-methods must have relatively different execution times. So the synchronization imposes the faster methods to wait for the slowest before restarting a cycle. The size of the information to share also becomes more important; hence the communications take more time.

We can conclude that a better execution time can be obtained when the behavior profilers are created in a high performance computing environment. However, the synchronizations in the algorithm limit those benefits if the computed number of events is more than 500000 events.

We can also conclude that the CN/SN 7 parallel programming approach presents overall better performance than a client-server approach. However, we can discuss the fact these last two *multithreading*(MT) and *multiprocessing*(MP) libraries are not built to work for multinode cluster architectures but more for multicore and settings.

11.4.2 Scalability test

Figure 11.2 presents the strong scalability of UCEL. We run this experiment using a multi-UCEL implementation with 10 MLP as co-methods and a fixed activity dataset size of 500000 entries. The result highlights that the speedup of the parallel behavior profiler rises from 1 to approximately 4.5 when we increase the number of processing cores. This figure demonstrates the strong scalability character of the UCEL implementation 7.

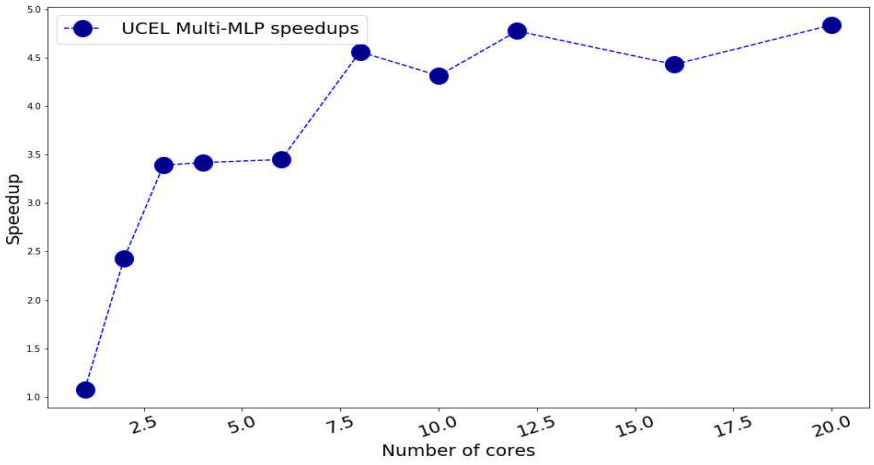

Figure 11.2: Parallel *behavior profiler* strong scalability on GRID'5K (10 co-methods)

11.5 Execution time

Here we attempt to determine the average execution time of each co method. For that purpose, we plot the scaled training and validation time of each co-methods for the same data of "1000 events". The results of this experiment are presented in figure 11.3. These results allow us to determine that the autoencoder, the PageRank, and the isolation forest are the methods that take the most training time. They also revealed that the autoencoder-based method takes almost ten times longer than the other co-methods.

The construction of the learner python class can explain that the autoencoder training is longer. Compared to the other co-methods, which are based on the *scikit-learn* library, the autoencoder is based on the *TensorFlow* library. Besides, we use multiple reshaping steps to turn the training and validation datasets in the form of a tensor.

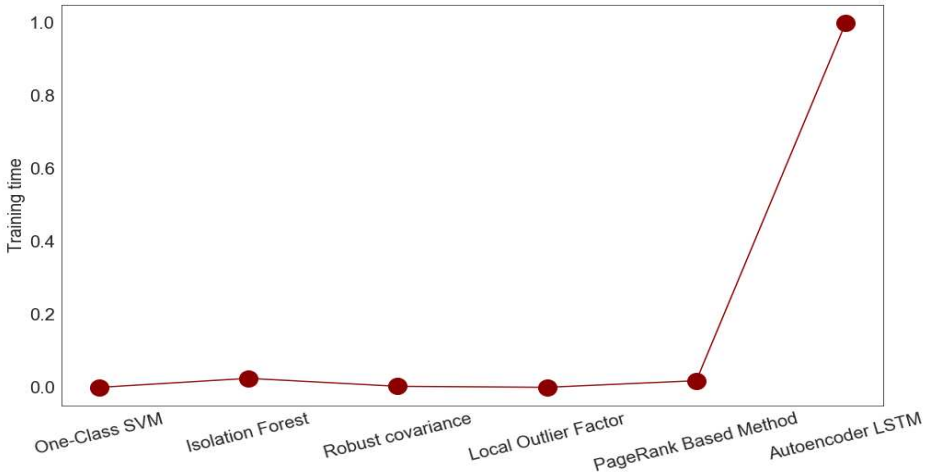

Figure 11.3: Execution time of co-methods

Overall, this experimentation taught us that the co-method execution times are not equal. The autoencoder-based method is one of the bottlenecks of the parallel behavior profiler algorithm we proposed.

However, with our experiment on G5K, we observed that the ratio is inverted when the size of the dataset is large enough, and we use multithreading. When the dataset is large enough ($size > 250000$ events), the multithreading capabilities of the autoencoder based on the *TensorFlow* library allows a faster computation than for the other co-methods. This makes its optimization very important to have a global performant UCEL framework.

11.6 First asynchronous tests

In order to remedy the bottleneck of synchronous communication, it is possible to use inter-node communications in fully asynchronous mode. This means that the communications of steps 5 and 7 of the algorithm 7 can be done by overlapping the other steps of the algorithm. In other words, in step 5 of the algorithm 7, the ith process sends its data from a server node to the control node whose task is to update the intermediate results of all the processes and to compute periodically the best of them (step 6 of the algorithm 7). After this sending and without waiting for the other nodes, the ith process continues its calculation task with the last available data received from the control process. We propose implementing the behavior profile based on a modification of the UCEL framework model that we present in this figure 11.4. We then test this model on the G5K.

In this experiments 11.6 and 11.5, we used the 8 co-methods (2-IForest, Robcov, 2-LOF, Ocsvm, PRBM, LSTM). We compared the *CN/SN*, parallel programming model to the asynchronous model.

From the test results 11.6 and 11.5, we can conclude that the asynchronous approach performs better than the synchronous approach. This programming model does not have synchronization steps.

This programming model does not have synchronization steps. The number of communications is reduced since

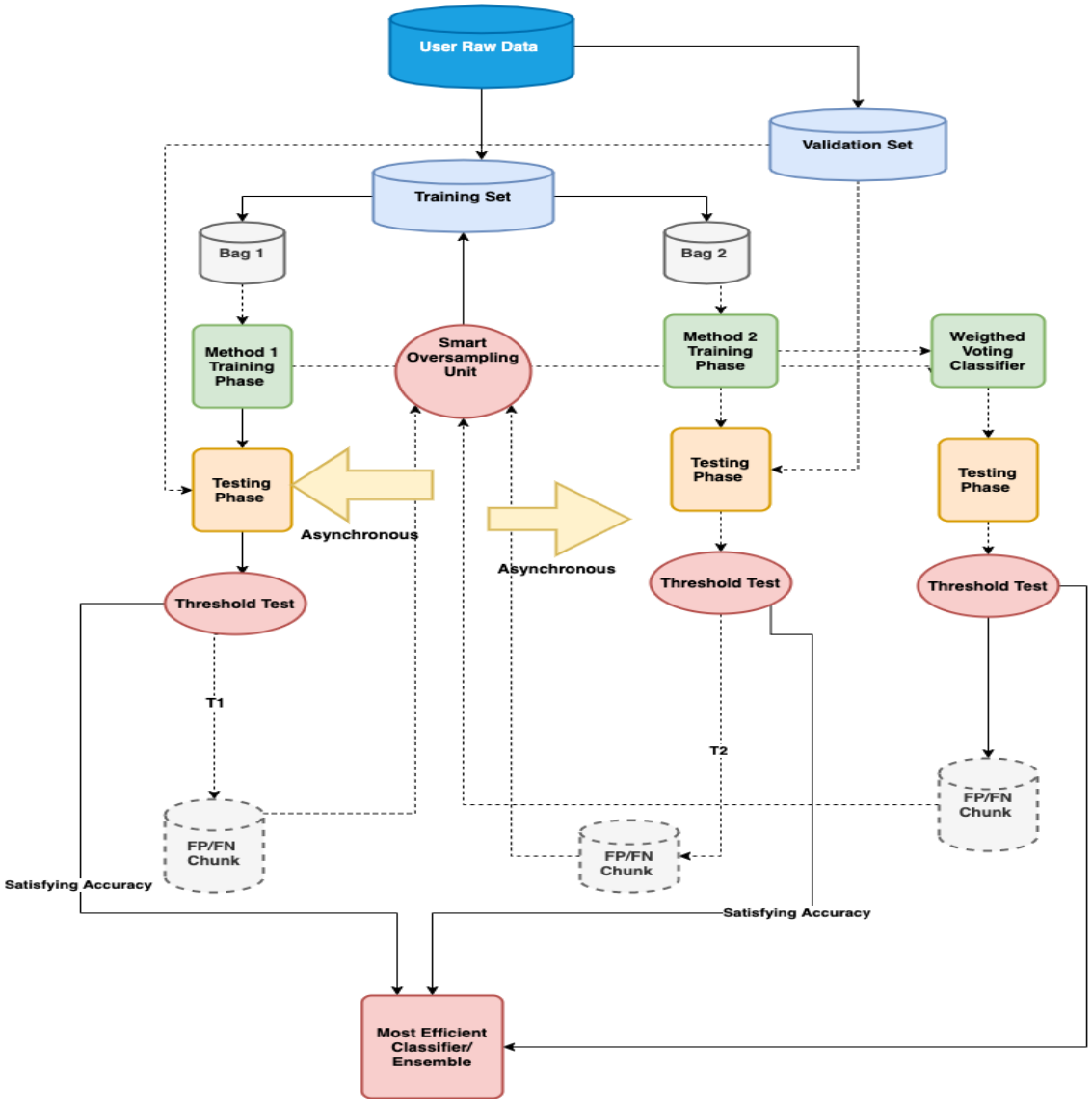

Figure 11.4: Asynchronous communication model

each co-method shares its results as soon as it finishes its computation. That means the update of intermediary results (their improvements) occurs more frequently than in the asynchronous case. Each computing node updates its training bags when it receives new *FP/FN* chunks. Hence, the co-methods continue their iteration with the information at their disposal.

The intermediate results of this model might be a bit different from the synchronous model, which does not start new iterations until all co-methods finish their training phases and send its *FP/FN*. However, in the end, the information is still shared between the co-methods after every training and testing cycle.

This experiment also showcases that the number of co-methods plays a role in the peak of the speedup value. We did a previous experiment with 6 co-methods and we obtained a max speedup of 3.2 for the asynchronous model. In the experiment 11.6 for eight co-methods we obtained 3.8, and for the first experiment with nine co-methods we

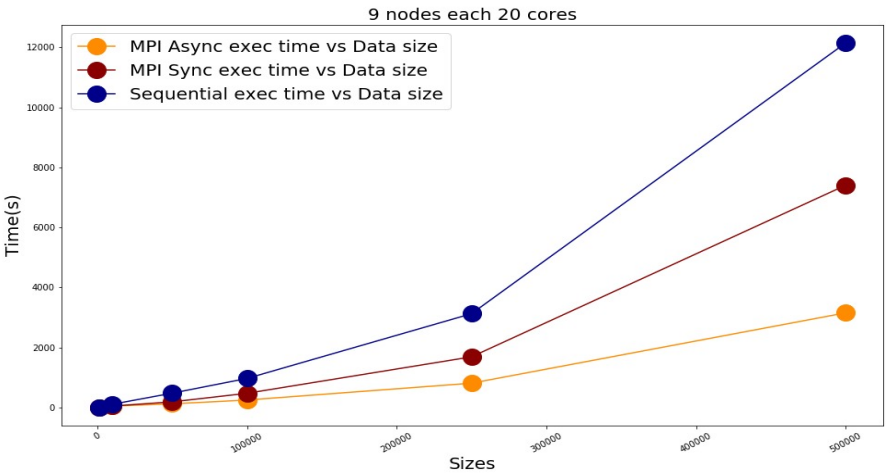

Figure 11.5: Asynchronous vs synchronous model execution time test

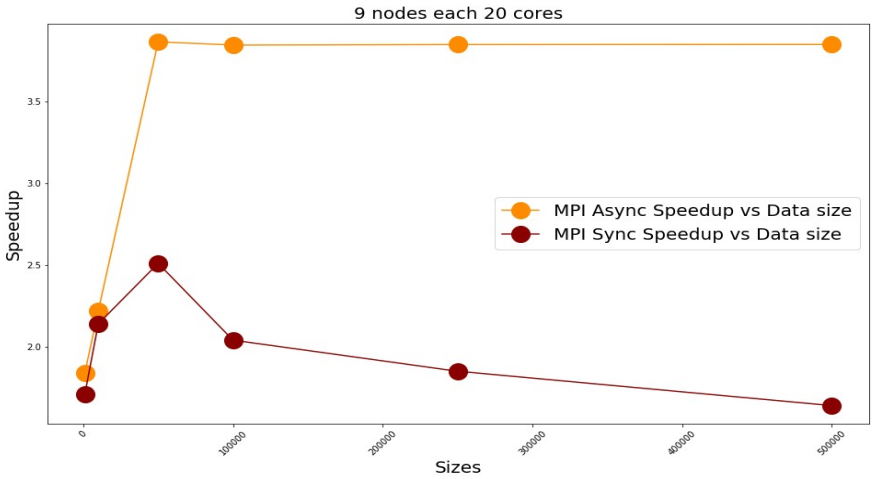

Figure 11.6: Asynchronous vs synchronous model speedup test

get a speedup of 4.45 11.1.

We plan to do more experimentation in future works, but for now, we can conclude that an asynchronous approach performs better than the synchronous model.

11.7 Handling multiple analysis of user

Especially for big companies, the number of users to monitor can be huge. High performance computing plays an essential role for that purpose. We test how a multi-graph-based UCEL, a multi auto-encoder model, and a classic UCELframework handle numerous users in this experimentation.

11.7.1 PageRank and Autoencoder based UCEL

In the figures 11.7, 11.8, 11.9, 11.10 we present the respectively the execution times and the speedups for the graph based methods and the speedups.

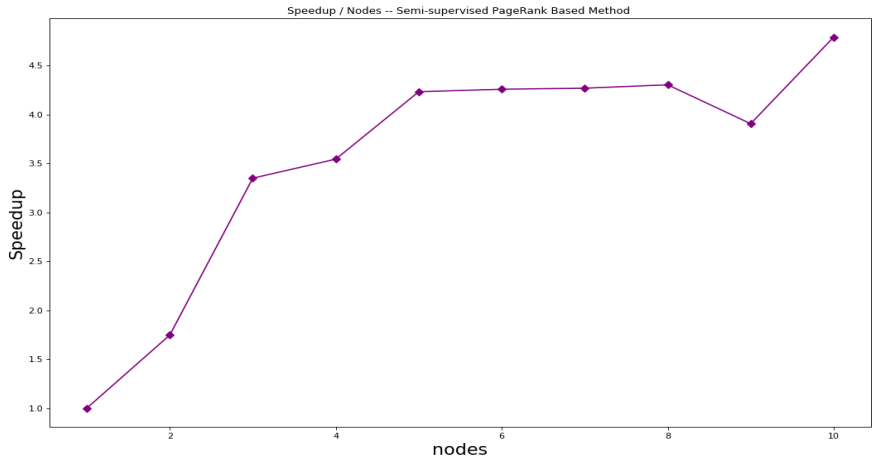

Figure 11.7: Speedup of PageRank based method, 10 nodes for 10 users

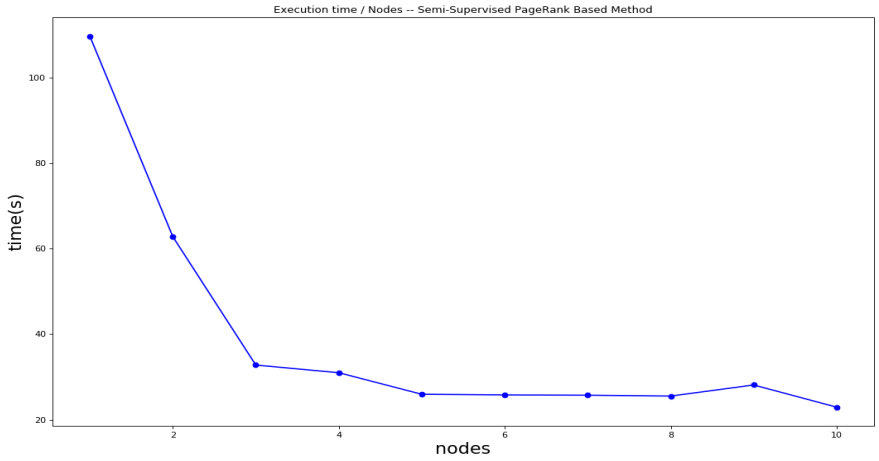

Figure 11.8: Execution time measure of PageRank based method, 10 nodes for 10 users

We use here an algorithm that splits the USERS in the function of the resources (i.e., the number of available nodes). It is based on the use of the *mpi4py* library the multithreading capabilities of each base method. This algorithm affects each user to a cluster node. In this case, each node works with an instance of the UCEL 7. These nodes also use the available cores with their multithreading capabilities of the co-methods.

To manage the multithreading, we use the *joblib* backend to parallelize the training for the graph-based method. For the autoencoder, we use the computation distribution strategies of the *TensorFlow* libraries. These strategies

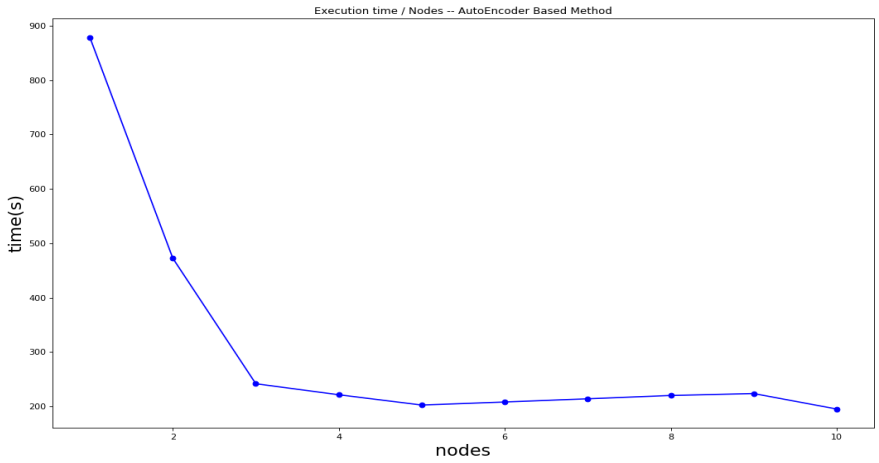

Figure 11.9: Time measure of autoencoder based method, 10 nodes for 10 users

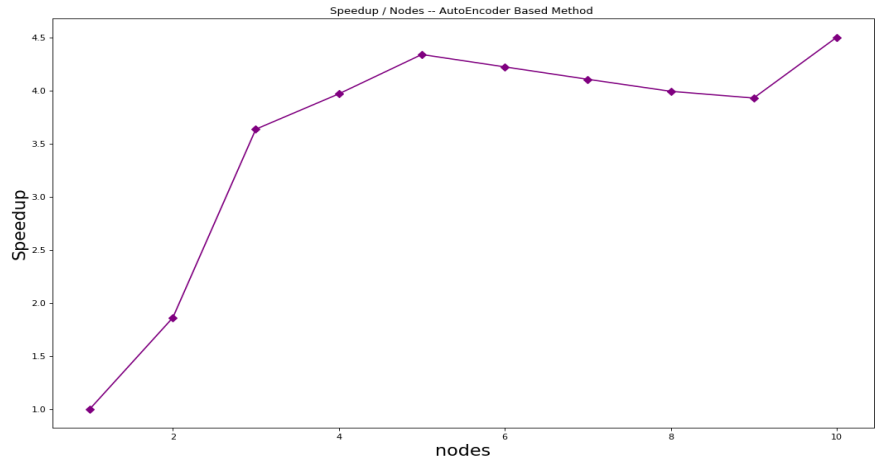

Figure 11.10: Speedup of autoencoder based method, 10 nodes for 10 users

use the different core available in the working nodes to distribute their calculations. In both cases, the trained results of the profilers are sent to a chosen node in the end. This last step helps store the result of the framework and prepare an eventual transfer of information to other cybersecurity tools.

Overall, these figures 11.7, 11.8, 11.9, 11.10 showcase that the training time of the behavior profiles diminishes if there are more resources to handles the users. The management of multiple users directly impacts the time necessary to protect the global informational assets of the companies. It showcases the usefulness of high-performance techniques and high-performance architectures for an insider threats detection system.

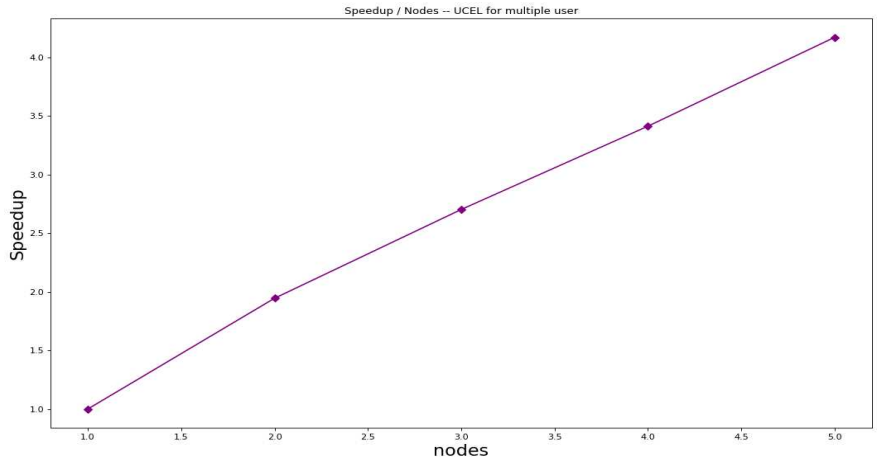

Figure 11.11: UCEL speedup handling 5 users

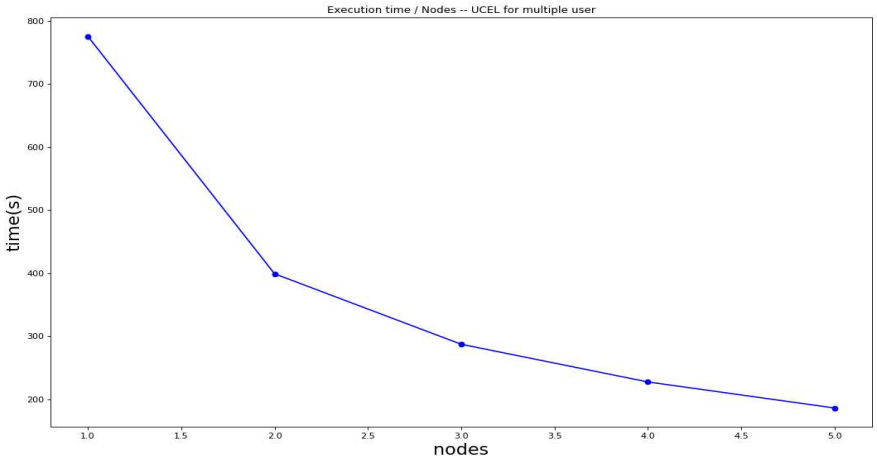

Figure 11.12: UCEL Time handling 5 users

11.7.2 UCEL with all co-methods and fewer ressources

In the figures 11.13, 11.14 we test multiple users. However, this time, the UCEL framework works with different co-methods, and we have access to fewer resources. In a real case study, this will probably be the case since the number of users in big companies is probably higher than the number of computation resources available.

For instance, we test the case where there are only five nodes with 20 users. This means that each of the five nodes handles that analysis of 4 users. In this case, we still obtain a speedup gain using a cluster architecture. This leverages the fact that the implementation of UCEL efficiently manages multiples users when it is run on architecture with fewer resources.

In figure 11.15, 11.16 we compare the speedup and execution time of the UCEL handling 5 to 50 users. We observe that even though the execution times are different, the speedup stays relatively the same ($=3$) if we have

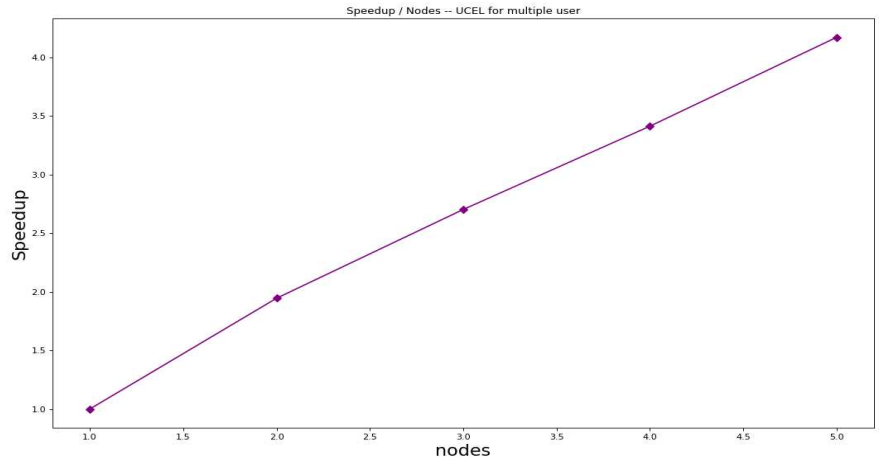

Figure 11.13: UCEL speedup handling 5 users

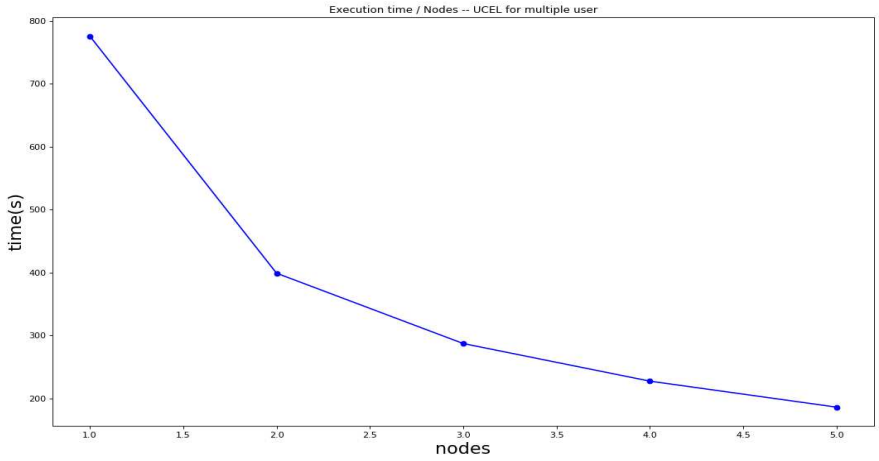

Figure 11.14: UCEL execution time handling 5 users

five nodes as the computation resources.

11.8 GPU impact test

We discussed in section 11.5 that the autoencoder is the method that takes more time. Since its classifier class is based on *TensorFlow*, we use its capabilities for task parallelization. To use the maximum possible configuration possible, we use a node with 40 cores associated with GPUs. Furthermore, we optimize the LSTM layers to use the *CUDNN* library to parallel this method using GPUs. These libraries allow parallelizing the computation of the artificial neural network using *Nvidia* GPUs and the *CUDA* libraries.

For that purpose, it was necessary to modify the configuration of the autoencoder-based methods. *CUDNN*

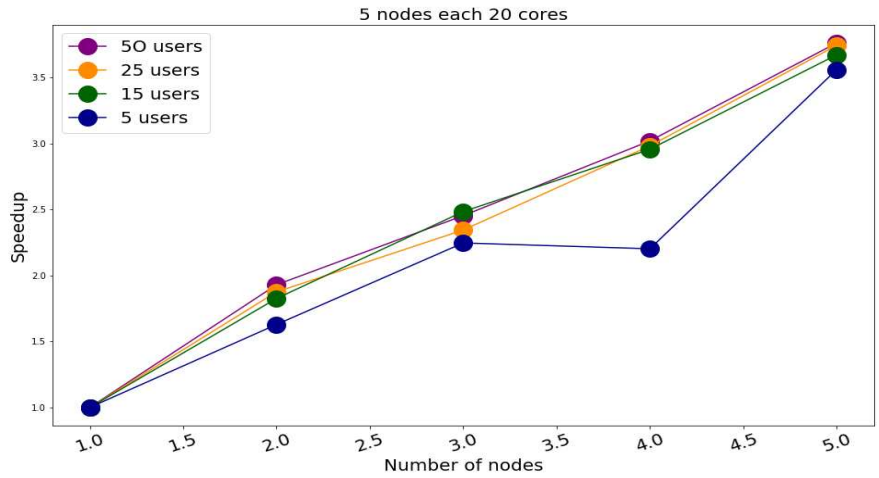

Figure 11.15: UCEL speedup handling 5 users

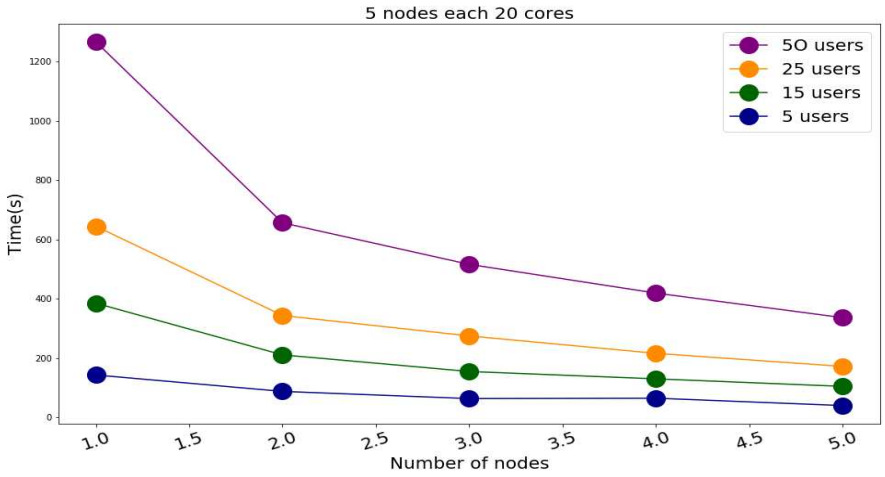

Figure 11.16: UCEL execution time handling 5 users

impose specifics layers and node configuration in order to send the computation of artificial neural networks to a GPU device. These configurations are the scope of the activations functions(e.g., *tanh*) and other hyperparameters choices.

We obtain a speedup up to 10 for the autoencoder methods with two GPUs(see figure 11.17). This is the best speedup we got, compared to the different parallel programming models. It is important to note that in this case, that we do not see performance improvement if the size of the data is not big. We did this test with a dataset of ten million events. With a smaller dataset size, the transfer of data to the GPU takes too much time to have a significant speedup gain. This is also dependant on the cluster architecture. Hence, the GPU allows good performance. However, only if there is sufficient activity data.

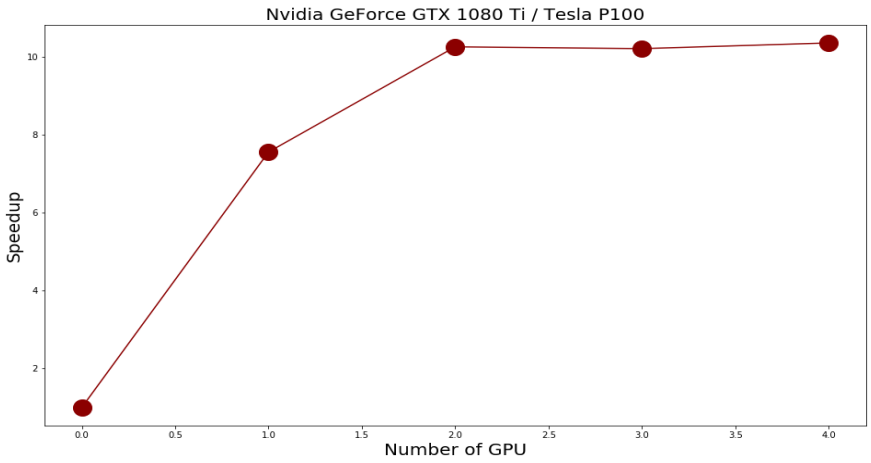

Figure 11.17: GPU Speedup

11.9 Conclusion

In this part, we did multiple tests of the proposed parallel programming models. One of the key elements of UCEL is its intrinsic parallelism allowing efficient implementation and therefore saving time and significant scaling. Its fault tolerance capabilities also allow to offer a solution in case one co-methods is performing in an inadequate manner. Using a parallel programming model with asynchronous communication, we obtained a speedup up to 3.8. We have also taken into account the parallelism inter co-methods in our implementations. As we have shown by the study of GPU impact, exploiting the parallelism intra co-methods in addition with that one of inter co-methods can impact the time very positively to a solution of large scale applications such as big data classification and cybersecurity problems. The best speedup test we obtained was with the GPU usage on the autoencoder method. Overall, these tests showcase the importance of the use of HPC techniques to have the best computation efficient insider threat detection systems.

Chapter 12

Conclusion

12.1 Summary of thesis

Insider threat detection is one of the main problems that cybersecurity experts face today. Depending on the security systems of the companies, insiders can do significant damage. Nowadays the user and entity behavior analysis is the method used to detect insider attacks. In this thesis, we presented UEBA methods based on the combination of techniques from the domains of cybersecurity, data science, and high-performance computing. Initially, the most used method against these issues was systems based on expert-written rules. However, nowadays, cybersecurity companies focus on using data science and machine learning techniques to build UEBA solutions.

To provide an effective response to this problem, we have proposed the UCEL framework based on the application of the unite and conquer approach to ensemble learning techniques against insider threat detection. We consider insider threat detection as a classification problem that addresses issues that classification methods face today.

We have shown that it is possible to look at this problem as a classification problem with a highly imbalanced dataset. It encouraged to use of anomaly or novelty detection methods. However, after studying the problem, we found the UEBA solution based on these methods suffers from a high rate of false-positive and false-negative.

We highlighted that the proposed framework, called UCEL, gives reliable results in an insider threats detection use case. This system can build user profiles capable of classifying new unlabeled activities as normal or abnormal. These ensemble approaches use base methods in the family of anomaly detection methods and classic supervised learning and graph-based techniques.

Indeed, we have shown that UCEL increases the performances of all weak learners, composing the behavior profiler. However, it does not significantly impact solid learners (i.e., initially performing well). But the presence of this latter, as co-methods, positively impacts the global method's results. We studied several insider attack scenarios, and we showed that the UCEL framework is reliable and gives good results. The study of these scenarios confirmed

that the best detection strategy is to adopt semi-supervised methods when the data is imbalanced and then to use supervised methods when the companies dispose of enough labeled data. We also proposed custom co-methods based on graph analysis and autoencoder combined with recurrent neural networks. We pointed out the interest in their integration into the UCEL framework.

In this thesis, insider threat is the focal point of the study. It should be noted that the UCEL approach is much more general and could be applied to all types of iterative data science methods. UCEL can cover a wide range of machine learning methods as core learners or co-methods. It will allow boosting their performances, especially in the case of the unbalanced dataset.

To study the performance of a parallel version of our behavior profiler, we used a kind of parallel client-server (simulated shared memory) implementation of the algorithm 7 on the *GRID5000* platform. The presented results show that we can obtain up to 4.45 speedup. Nevertheless, due to a synchronization step after each cycle, this speedup decreases when the dataset size increases. The solution to this issue is to use asynchronous communications in the algorithm. The first experiments we did for asynchronous is promising with a speedup that peaks at 3.8. This approach outperforms the synchronous approach in the same conditions (peak 2.5).

We also used other high performance computing strategies. One focuses on the share of the users analyzed, and the other using GPUs. Overall, the max speed up obtained for UCEL is 10 with the GPU. However, the use of GPU makes sense only if the number of a user is consequent. This might be well-adapted to a role-based insider threat detection software based on the role profile, where the analyzed data are significant.

The UCEL framework is integrated into the global insider threat module based on the container architecture. We built this system that adds to the UCEL framework an exploratory data analysis blocs, a preprocessing, and an alarm system in containers. The alarm systems linked to this module showed promising results. It computes risk scores that signal the insider attack and can be used by human operators, other cybersecurity tools, or launched contextual decisions. The goal is to link this module as an extension of IAM software or other cybersecurity tools. Overall, this combination of containers allows us to build a scalable machine learning pipeline capable of detecting insider attacks in real-time.

12.2 Some perspectives and future works

Today we use the data from the Carnegie melon university to build our insider threat detection systems. One of our goals was to implement custom solutions that extend an IAM solution. Since we only dispose of sparse IAM audit data, our future will be to build a simulator of IAM activity that follows logical user behavior. The goal is to construct normal activities and attack scenarios close to real-life situations.

Hence, after dealing with an insider threats module for IAM tools using a coherent user behavior simulator, we want to focus on other data sources such as third-party log applications. This new data flow will allow building more

models for more diverse attack scenarios.

It is also possible to improve the prepossessing step with additional approaches to get session characteristics, use decoy files or build an email network to have additional strategies to detect attacks. It is also possible to choose new strategies to select the user activity subset that allows comparing the behavior of the users to their organization peers.

As it is currently constructed, the UCEL framework works with two oversampling techniques (random and synthetic minority oversampling). In the future, we want to investigate other oversampling methods to improve the collaboration systems between co-methods. Sharing the hyperparameters during the iteration also seems to be a way we can improve our UCEL process. Today even if we start with the best hyperparameters, during the iteration process, the data composition changes. Hence the best hyperparameters might also be changes. We can also use the information of the PageRank of the data samples as a new feature. We did some preliminary experiments, but we didn't get satisfactory results. However, it is a direction we want to follow.

If we focus on our parallel implementation, it will be interesting to investigate other programming models. Communications are one of the most limiting factors of solutions. It is possible to investigate the option of storing information on the file systems to avoid communication between iterations. Today the best UCEL algorithm uses the main message passing interface (MPI) to manage the communication between nodes and to send the built behavior profile. It will be interesting to use other communication libraries use and evaluate their performance.

The final purpose of this work is to build an industrial module capable of working on a cloud or on-premise environment capable of detecting insider threats. For now, we are at the beginning of the industrialization process. Our solution works on virtual machines with the Docker software. As a perspective, we would like to use the *Kubernetes* and Kubeflow platforms to deploy our solution easily on a different platform. This will allow building a versatile and portable framework using a platform increasingly adopted by the industries in relation to our insider threat detection systems. Kubernetes is a container orchestrator that also manages its scaling on distributed architectures. Kubeflow is a Kubernetes toolkit that allows building a machine learning pipeline of tests easily deployable. We also envision adding the connexion to an extreme gradient boosting framework to this platform.

For the alarm systems, it would be interesting to choose other time splits to build more detection systems that cover the detection on more considerable periods and more types of attacks. Now the system monitors daily activities. An extension to week and month activities could add an extra layer of protection.

We can also use other parallel computing strategies. One involves the distribution of the computation of the containers using the scalability capabilities of Kubernetes. Another that exploits the GPU parallelization of the methods since the autoencoder can profit from this type of device. Ultimately we would like to propose an efficient parallel implementation of the detecting module, taking into account the asynchronicity of communications of the UCEL framework and capable of handling vast quantities of user data.

12.3 Ethics reflection on data protection

When the term user behavior analysis is used and explained, it always raises questions about the ethics of the methods. Monitoring user behavior can be perceived as a way to keep watch of their activities and to violate their privacy. These concerns are opposite to the goal of behavior analysis in cybersecurity. It is used in order to protect personal data, and in a way, user privacy.

Behavior analysis is based on intelligent algorithms. It is not the first time criticisms target the use of artificial intelligence for its possible drifts and misuses. The usage of this advanced technology is not yet in the phase of acceptance that other security technologies had to reach (i.e., the implementation of city cameras as a dissuasive and a surveillance tool). They were not accepted directly by public opinion, but they helped to identify true criminals, increase the security of areas, and reduce the criminally around.

The main goal of user behavior analysis should not be misunderstood. It is first of all to stop true threats and to protect users against hackers capable to steal their credentials (i.e., masqueraders). This last individual can perpetrate malicious actions under their name. Honest employees could face disastrous consequences and accusations if their access rights are misused or personal information stolen.

UEBA tools use advanced alerting systems based on security thresholds. They can generate alerts or contextual decisions. The response to the detection of an abnormal behavior depends on thresholds. For example, some inadvertent actions could be considered are harmless or trigger simple email alerts to users.

In this thesis, we established that the only way to counter intruders efficiently, insiders, and fraudsters is through the use of user behavior analysis. We strongly believe that a correct implementation (i.e., one that will have to consider some ethical rules) of these algorithms would brush off all the privacy concerns that are associated with this kind of tool. The danger that represents insider threat menace is too great to deal with without monitoring methods, especially if this benefits end-users.

Appendix A

First Appendix

Appendix B

Second Appendix

Bibliography

[1] L. Akoglu, M. McGlohon, and C. Faloutsos. Oddball: Spotting anomalies in weighted graphs. In *Pacific-Asia conference on knowledge discovery and data mining*, pages 410–421. Springer, 2010.

[2] D. Ayres-de Campos, J. Bernardes, A. Garrido, J. Marques-de Sa, and L. Pereira-Leite. Sisporto 2.0: a program for automated analysis of cardiotocograms. *Journal of Maternal-Fetal Medicine*, 9(5):311–318, 2000.

[3] A. Beutel, W. Xu, V. Guruswami, C. Palow, and C. Faloutsos. Copycatch: stopping group attacks by spotting lockstep behavior in social networks. In *Proceedings of the 22nd international conference on World Wide Web*, pages 119–130, 2013.

[4] A. Beutel, L. Akoglu, and C. Faloutsos. Graph-based user behavior modeling: from prediction to fraud detection. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 2309–2310, 2015.

[5] T. Bonald, N. de Lara, Q. Lutz, and B. Charpentier. Scikit-network: Graph analysis in python. *Journal of Machine Learning Research*, 21(185):1–6, 2020.

[6] B. Bowen, M. B. Salem, S. Hershkop, A. Keromytis, and S. Stolfo. Designing host and network sensors to mitigate the insider threat. *IEEE security & privacy*, 7(6):22–29, 2009.

[7] L. Breiman. Pasting small votes for classification in large databases and on-line. *Machine learning*, 36(1-2): 85–103, 1999.

[8] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[9] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[10] A. L. Buczak and E. Guven. A survey of data mining and machine learning methods for cyber security intrusion detection. *IEEE Communications surveys & tutorials*, 18(2):1153–1176, 2015.

[11] P. Casas, J. Mazel, and P. Owezarski. Unsupervised network intrusion detection systems: Detecting the unknown without knowledge. *Computer Communications*, 35(7):772–783, 2012.

[12] X.-w. Chen and J. C. Jeong. Enhanced recursive feature elimination. In *Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*, pages 429–435. IEEE, 2007.

[13] Y. Chen, S. Nyemba, W. Zhang, and B. Malin. Specializing network analysis to detect anomalous insider actions. *Security informatics*, 1(1):5, 2012.

[14] T. Chengsheng, L. Huacheng, and X. Bing. Adaboost typical algorithm and its application research. In *MATEC Web of Conferences*, volume 139, page 00222. EDP Sciences, 2017.

[15] L. Choy, O. Delannoy, N. Emad, and S. G. Petiton. Federation and abstraction of heterogeneous global computing platforms with the yml framework. In *2009 International Conference on Complex, Intelligent and Software Intensive Systems*, pages 451–456. IEEE, 2009.

[16] A. Chuvakin and A. Barros. A comparison of ueba technologies and solutions. Technical report, Gartner, 2017.

[17] C. Cortes, D. Pregibon, and C. Volinsky. Communities of interest. In *International Symposium on Intelligent Data Analysis*, pages 105–114. Springer, 2001.

[18] A. Dal Pozzolo, O. Caelen, R. A. Johnson, and G. Bontempi. Calibrating probability with undersampling for unbalanced classification. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 159–166. IEEE, 2015.

[19] A. Dal Pozzolo, G. Boracchi, O. Caelen, C. Alippi, and G. Bontempi. Credit card fraud detection: a realistic modeling and a novel learning strategy. *IEEE transactions on neural networks and learning systems*, 29(8): 3784–3797, 2017.

[20] D. Davison and W. Baoning. Propagating trust and distrust to demote web spam, 2006.

[21] A. Diop, N. Emad, T. Winter, and M. Hilia. Design of an ensemble learning behavior anomaly detection framework. *International Journal of Computer and Information Engineering*, 13(10):551–559, 2019.

[22] A. Diop, N. Emad, and T. Winter. A parallel and scalable framework for insiderthreat detection. In *2020 IEEE 27th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, pages 101–110. IEEE, 2020.

[23] A. Diop, N. Emad, and T. Winter. A unite and conquer based ensemble learning method for user behavior modeling. In *2020 IEEE International Performance Computing and Communications Conference*, page 4. IEEE, 2020.

[24] D. Dua and C. Graff. UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

[25] W. Eberle, J. Graves, and L. Holder. Insider threat detection using a graph-based approach. *Journal of Applied Security Research*, 6(1):32–81, 2010.

[26] E. Ekinci, N. Acun, et al. A comparative study on machine learning techniques using titanic dataset. In *7th international conference on advanced technologies*, pages 411–416, 2018.

[27] N. Emad and S. Petiton. Unite and conquer approach for high scale numerical computing. *Journal of computational science*, 14:5–14, 2016.

[28] N. Emad, S.-A. Shahzadeh-Fazeli, and J. Dongarra. An asynchronous algorithm on the netsolve global computing system. *Future Generation Computer Systems*, 22(3):279–290, 2006.

[29] N. Emad, L. Drummond, M. Tsuji, and M. Dandouna. Tuning asynchronous co-methods for large-scale eigenvalue calculations. In *16th SIAM Conference on Parallel Processing for Scientific Computing*, page 8. SIAM, 2014.

[30] H. P. Entreprises. Ciso guide to machine learning and user behavior analytics. Technical report, ciso, 2018.

[31] K. Fauvel, V. Masson, E. Fromont, P. Faverdin, and A. Termier. Towards sustainable dairy management-a machine learning enhanced method for estrus detection. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3051–3059, 2019.

[32] E. Fix. *Discriminatory analysis: nonparametric discrimination, consistency properties*. USAF School of Aviation Medicine, 1951.

[33] A. Gamachchi, L. Sun, and S. Boztas. A graph based framework for malicious insider threat detection. In *50th Hawaii International Conference on System Sciences*, 2017.

[34] A. Geron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Sebastopol, CA, 2017. ISBN 978-1491962299.

[35] J. Glasser and B. Lindauer. Bridging the gap: A pragmatic approach to generating insider threat data. In *2013 IEEE Security and Privacy Workshops*, pages 98–104. IEEE, 2013.

[36] N. Görnitz, M. Kloft, K. Rieck, and U. Brefeld. Toward supervised anomaly detection. *Journal of Artificial Intelligence Research*, 46:235–262, 2013.

[37] Gurucul. User and entity behavior analysis use cases. Technical report, Gurucul, 2016.

[38] Z. Gyongyi, H. Garcia-Molina, and J. Pedersen. Combating web spam with trustrank. In *Proceedings of the 30th international conference on very large data bases (VLDB)*, 2004.

[39] D. Haidar and M. M. Gaber. Adaptive one-class ensemble-based anomaly detection: an application to insider threats. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–9. IEEE, 2018.

[40] A. J. Hall, N. Pitropakis, W. J. Buchanan, and N. Moradpoor. Predicting malicious insider threat scenarios using organizational data and a heterogeneous stack-classifier. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 5034–5039. IEEE, 2018.

[41] H. He, G. Bergère, and S. Petiton. A hybrid gmres/ls-arnoldi method to accelerate the parallel solution of linear systems. *Computers & Mathematics with Applications*, 51(11):1647–1662, 2006.

[42] N. Hu, P. G. Bradford, and J. Liu. Applying role based access control and genetic algorithms to insider threat detection. In *Proceedings of the 44th annual Southeast regional conference*, pages 790–791, 2006.

[43] IBM-Security. Specializing network analysis to detect anomalous insider actions. *Managed Security services*, https://securityintelligence.com/media/cyber-security-intelligence-index-2015/, 2016.

[44] C. Insiders and Gurucul. Insider threat report 2020. Technical report, Cybersecurity Insiders, 2019.

[45] A. H. Jahromi and M. Taheri. A non-parametric mixture of gaussian naive bayes classifiers based on local independent features. In *2017 Artificial Intelligence and Signal Processing Conference (AISP)*, pages 209–212. IEEE, 2017.

[46] M. Jiang, P. Cui, A. Beutel, C. Faloutsos, and S. Yang. Catchsync: catching synchronized behavior in large directed graphs. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 941–950, 2014.

[47] X. Jin, A. Xu, R. Bie, and P. Guo. Machine learning techniques and chi-square feature selection for cancer classification using sage gene expression profiles. In *International Workshop on Data Mining for Biomedical Applications*, pages 106–115. Springer, 2006.

[48] A. Khwaja, A. Anpalagan, M. Naeem, and B. Venkatesh. Joint bagged-boosted artificial neural networks: Using ensemble machine learning to improve short-term electricity load forecasting. *Electric Power Systems Research*, 179:106080, 2020.

[49] J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[50] Y. Koren. Collaborative filtering with temporal dynamics. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 447–456, 2009.

[51] S. Kotsiantis and P. Pintelas. Combining bagging and boosting. *International Journal of Computational Intelligence*, 1(4):324–333, 2004.

[52] X. Li, Y. Xue, and B. Malin. Detecting anomalous user behaviors in workflow-driven web applications. In *2012 IEEE 31st Symposium on Reliable Distributed Systems*, pages 1–10. IEEE, 2012.

[53] Z. Li, H. Xiong, and Y. Liu. Detecting blackholes and volcanoes in directed networks. *arXiv preprint arXiv:1005.2179*, 2010.

[54] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, 2008.

[55] S. Masoudnia and R. Ebrahimpour. Mixture of experts: a literature survey. *Artificial Intelligence Review*, 42(2): 275–293, 2014.

[56] R. A. Maxion and T. N. Townsend. Masquerade detection using truncated command lines. In *Proceedings international conference on dependable systems and networks*, pages 219–228. IEEE, 2002.

[57] P. Moriano, J. Pendleton, S. Rich, and L. J. Camp. Stopping the insider at the gates: Protecting organizational assets through graph mining. *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.*, 9(1):4–29, 2018.

[58] C. C. Noble and D. J. Cook. Graph-based anomaly detection. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636, 2003.

[59] P. Parveen, N. Mcdaniel, Z. Weger, J. Evans, B. Thuraisingham, K. Hamlen, and L. Khan. Evolving insider threat detection stream mining perspective. *International Journal on Artificial Intelligence Tools*, 22(05):1360013, 2013.

[60] D. Peña and F. J. Prieto. Multivariate outlier detection and robust covariance matrix estimation. *Technometrics*, 43(3):286–310, 2001.

[61] J. Peng, K.-K. R. Choo, and H. Ashman. User profiling in intrusion detection: A review. *Journal of Network and Computer Applications*, 72:14–27, 2016.

[62] L. E. Peterson. K-nearest neighbor. *Scholarpedia*, 4(2):1883, 2009.

[63] Ponemon. 2018 coast of insider threat global organizations, ponemon insitute research report. *Managed Security services*, https://www.observeit.com/ponemon-report-cost-of-insider-threats/(Last accessed 4), 2018.

[64] M.-C. Popescu, V. E. Balas, L. Perescu-Popescu, and N. Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009.

[65] Y. Saeys, T. Abeel, and Y. Van de Peer. Robust feature selection using ensemble feature selection techniques. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 313–325. Springer, 2008.

[66] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, and J. C. Platt. Support vector method for novelty detection. In *Advances in neural information processing systems*, pages 582–588, 2000.

[67] M. Schonlau and M. Theus. Detecting masquerades in intrusion detection based on unpopular commands. *Information Processing Letters*, 76(1-2):33–38, 2000.

[68] M. Schonlau, W. DuMouchel, W.-H. Ju, A. F. Karr, M. Theus, and Y. Vardi. Computer intrusion: Detecting masquerades. *Statistical science*, pages 58–74, 2001.

[69] M. Sebban, R. Nock, S. Lallich, et al. Stopping criterion for boosting-based data reduction techniques: From binary to multiclass problem. *J. Mach. Learn. Res.*, 3:863–885, 2002.

[70] N. M. Sheykhkanloo and A. Hall. Insider threat detection using supervised machine learning algorithms on an extremely imbalanced dataset. *International Journal of Cyber Warfare and Terrorism (IJCWT)*, 10(2):1–26, 2020.

[71] M. Sikora et al. Application of rule induction algorithms for analysis of data collected by seismic hazard monitoring systems in coal mines. *Archives of Mining Sciences*, 55(1):91–114, 2010.

[72] L. Sun, S. Versteeg, S. Boztas, and A. Rao. Detecting anomalous user behavior using an extended isolation forest algorithm: an enterprise case study. *In Computer Research Repository(CoRR)*, 2016.

[73] P. P. Talukdar and K. Crammer. New regularized algorithms for transductive learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 442–457. Springer, 2009.

[74] P.-N. Tan, M. Steinbach, and V. Kumar. Data mining cluster analysis: basic concepts and algorithms. *Introduction to data mining*, pages 487–533, 2013.

[75] A. Tharwat. Linear vs. quadratic discriminant analysis classifier: a tutorial. *International Journal of Applied Pattern Recognition*, 3(2):145–180, 2016.

[76] A. Tuor, S. Kaplan, B. Hutchinson, N. Nichols, and S. Robinson. Deep learning for unsupervised insider threat detection in structured cybersecurity data streams. *The Workshops of the The Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

[77] J. Weston, S. Mukherjee, O. Chapelle, M. Pontil, T. Poggio, and V. Vapnik. Feature selection for svms. In *Advances in neural information processing systems*, pages 668–674, 2001.

[78] H. Xiao, B. Biggio, G. Brown, G. Fumera, C. Eckert, and F. Roli. Is feature selection secure against training data poisoning? In *International Conference on Machine Learning*, pages 1689–1698. PMLR, 2015.

[79] Z. Yao, P. Mark, and M. Rabbat. Anomaly detection using proximity graph and pagerank algorithm. *IEEE Transactions on Information Forensics and Security*, 7(4):1288–1300, 2012.

[80] F. Yuan, Y. Cao, Y. Shang, Y. Liu, J. Tan, and B. Fang. Insider threat detection with deep neural network. In *International Conference on Computational Science*, pages 43–54. Springer, 2018.

[81] C. Zhang and Y. Ma. *Ensemble machine learning: methods and applications*. Springer, 2012.

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Analyse haute performance de masses de données ; application à la détection d'anomalie dans le contexte de la gestion d'identité et d'accès

Mots clés : Menace interne, Science de données, Unite and conquer, Apprentissage d'ensemble, Calcul intensif, PageRank

Résumé : La protection des données est une question essentielle en matière de cybersécurité. Les organisations utilisent les logiciels de gestion des identités et des accès et les outils de cybersécurité traditionnels pour protéger leurs actifs informationnels contre les menaces externes. Cependant, elles manquent le plus souvent de solutions pour contrer les menaces internes provenant principalement des personnes ayant un accès légitime aux systèmes d'information de l'entreprise. Ce type de menaces est aujourd'hui la principale préoccupation des spécialistes de la cybersécurité. Les logiciels d'analyse du comportement des utilisateurs et des entités sont les outils utilisés par les cyber-spécialistes pour contrer efficacement les menaces internes. Cependant, les solutions existantes peuvent présenter des problèmes tels qu'un nombre élevé de fausse alarme, et un temps de préparation des modèles de détection conséquent quand les données d'activités sont de gros volumes. L'objectif de cette thèse est de contribuer à remédier à ces problèmes par la proposition d'une solution algorithmique et sa mise en œuvre efficace pour les architectures haute performance. Plus particulièrement, nous proposons une méthode de détection qui construit des profileurs de comportement en utilisant des techniques issues des domaines de l'apprentissage automatique, de l'algèbre linéaire et du calcul haute performance. Cette méthode est définie par l'application de l'approche "unir et conquérir" utilisée en algèbre linéaire, aux techniques d'apprentissage d'ensemble. En plus des méthodes d'apprentissage de base classiques, nous intégrons des méthodes innovantes de type PageRank et auto-encodeurs dans la méthode globale proposée. Cette nouvelle méthode de détection des menaces internes montre, selon nos expérimentations, une efficacité en termes de précision, allant jusqu'à 98% d'AUC. Ceci marque une augmentation significative par rapport aux méthodes de bases. Nous proposons aussi une mise en œuvre de cette méthode selon plusieurs paradigmes de programmation parallèle permettant d'obtenir des accélérations jusqu'au 10.
Nous avons intégré cette plateforme logicielle agrémentée de moyens de prétraitement de données, et d'un système d'alarme dans un module global de détection d'attaque internes, capable d'étendre des outils de cybersécurité.

Title : High performance big data analysis; application to anomaly detection in the context of identity and access management

Keywords : Insider threat, Data science, Unite and conquer, High performance computing, Ensemble learning, PageRank

Abstract : Data protection is a critical issue in cybersecurity. Organizations use identity and access management software and traditional cybersecurity tools to protect their information assets from external threats. However, they most often lack solutions to counter insider threats from individuals with legitimate access to corporate information systems. This type of threat is now the primary concern of cybersecurity specialists. User and entity behavior analysis software are the tools used by cyber specialists to counter insider threats effectively. However, existing solutions can present problems such as many false alarms and a consequent development time of detection models when the activity data is of large volumes.
This thesis aims to remedy these problems by proposing an algorithmic solution and its efficient implementation for high performance architectures. More precisely, we propose a detection method that builds behavioral profilers using techniques from the fields of machine learning, linear algebra and high performance computing. This method is defined by application of "unite and conquer" approach, used in linear algebra, to ensemble learning techniques. We integrate innovative methods of PageRank and autoencoder in the proposed ensemble method in addition to the classical basic machine learning methods.
According to our experiments, this new method of insider threat detection shows an average efficiency in terms of detection accuracy, up to 98% of AUC. This is a significant increase compared to base methods. We also propose an implementation of this method according to several parallel programming paradigms allowing us to obtain a speedup up to 10.
We have integrated this software platform with data preprocessing means and an alarm system into a global module for insider threat detection, capable of extending cybersecurity tools.

Maison du doctorat de l'Université Paris-Saclay
2ème étage aile ouest, Ecole normale supérieure Paris-Saclay
4 avenue des Sciences,
91190 Gif sur Yvette, France