Nuno André Cardoso

USER BEHAVIOR ANALYTICS IN THE CONTACT CENTER
INSIDER THREAT ASSESSMENT AND FRAUD DETECTION

UNIVERSIDADE Ɖ
COIMBRA

1 2 9 0

1 2 9 0

UNIVERSIDADE Ɖ
COIMBRA

Nuno André de Matos Lopes Cardoso

USER BEHAVIOR ANALYTICS IN THE CONTACT CENTER
INSIDER THREAT ASSESSMENT AND FRAUD DETECTION

**Dissertation in the context of the Master in Informatics Security,
advised by Professor Marco Vieira and presented to
Faculty of Sciences and Technology / Department of Informatics Engineering.**

July 2021

Faculty of Sciences and Technology

Department of Informatics Engineering

User behavior analytics in the contact center

Insider threat assessment and fraud detection

Nuno André de Matos Lopes Cardoso

Dissertation in the context of the Master in Informatics Security,
advised by Prof. Marco Vieira and presented to
the Faculty of Sciences and Technology / Department of Informatics Engineering.

July 2021

1 2 9 0

UNIVERSIDADE Ð
COIMBRA

This page is intentionally left blank.

Acknowledgments

First of all, I really appreciate all the help given by my advisor at Talkdesk, Pedro Rodrigues. It is hard to express de patience and all the helpful advice Pedro gave me during this period, and for that I am really grateful.

I would also like to thank my supervisor from DEI, professor Marco Vieira, for all the availability to help and excellent follow-up of the work being done.

Furthermore, I would like to say a big thank you for professor Nuno Antunes, José Flora and José Diogo, for all their importance in my academic path, patience and help throughout all these years.

I am grateful for my friends and colleagues at Talkdesk, namely Pedro Neves and Daniel Antunes, as well as Jorge Vieira for giving me the opportunity to do this work along with my activities at Talkdesk, and also the entire Guardian team for creating a great environment to work in.

Last but not least, I would like to thank my family and friends, for keeping me motivated, believing in me, and being there for me no matter what.

This page is intentionally left blank.

Abstract

With the continuous growth of cyber-crime in the past few years, the need for innovative and effective cybersecurity strategies is fundamental for every organization. Founded in 2011, Talkdesk is building a cloud-based contact-center product, which reached more than 1800 customers in 2020. Contact-center jobs are known to be precarious and with high turnover rates and, along with the ongoing trend of working from home posed by the pandemic of COVID-19, there is an increased risk concerning the likelihood of fraud by an insider actor (in the case, a contact-center agent).

Typical cybersecurity controls adopted by organizations, such as security information event management (SIEM), tend to focus on external threats, using rule-based matching mechanisms to create alerts on potential incidents. However, such mechanisms are unable to detect novel threat scenarios, reason why new approaches are necessary, such as those utilising artificial intelligence. User and entity behavior analytics (UEBA) is an emerging solution to complement the security controls of an organization, which leverages machine learning and, more specifically, anomaly detection, to create baselines of normal behavior of users or entities and attempts to detect significant deviations from those baselines, which could represent threats.

The main goal of this work consists in assessing the viability of using a UEBA framework to protect the customers of Talkdesk, with which contact-center staff would be monitored with the objective of detecting threats related to sensitive information theft, cyber fraud, insider abuse, and others. To achieve this, we started by exploring the most appropriate raw data sources (for UEBA) available and defining threat scenarios to tackle, followed by obtaining the data and implementing transformation pipelines to convert raw data into a format suitable for anomaly detection. We proceeded with exploratory data analysis and feature engineering and decided on the appropriate anomaly detection algorithms to evaluate and the validation strategy to use. We created a labeled dataset with domain knowledge expertise, defined several settings to vary (baseline period, feature set, contamination), and applied them with 5 different algorithms. We evaluated each combination of the settings defined using supervised classification metrics, with the autoencoder and principal components analysis (PCA) achieving the highest F1-scores: 0.97 and 0.95, respectively. We also evaluated 3 different interpretability methods, to explain the anomalies reported.

Finally, we deployed the framework using PCA and 1 month of real data from 2 clients, in a staging environment, registering a total of 76 anomalies incurred by 45 different agents, with 67 true positives and 9 false positives. We believe the framework is ready for production, requiring minor adjustments on the interpretability algorithm and a strategy to suppress anomalies similar to past ones reported as false positives through feedback from the clients.

Keywords

UEBA, Artificial intelligence, Machine learning, Cybersecurity, Anomaly detection, Fraud detection

This page is intentionally left blank.

Resumo

Com o crescimento contínuo do cibercrime em anos recentes, a necessidade de estratégias de cibersegurança inovadoras e eficazes tornam-se cada vez mais importantes para todo o tipo de organizações. Fundada em 2011, a Talkdesk está a desenvolver um produto de *contact-center* na nuvem, que chegou aos 1800 clientes em 2020. As carreiras em *contact-center* são conhecidas pela precariedade e alta rotatividade de agentes que, agravada pela tendência de trabalho remoto imposta pela pandemia de COVID-19, aumenta o risco de ocorrência de fraude.

As abordagens de cibersegurança tipicamente adotadas pelas organizações, como gestão e correlação de eventos de segurança (SIEM), têm o seu foco em ameaças externas, através de um motor de regras que resulta em alertas indicativos de potenciais incidentes. No entanto, esses mecanismos têm a limitação da incapacidade de detetar modelos de ameaça inovadores, razão pelo qual abordagens que utilizam inteligência artificial são necessárias. A análise comportamental de utilizadores e entidades (UEBA) é uma solução emergente utilizada para complementar as estratégias de segurança de uma organização, que utiliza *machine learning* e, mais especificamente, algoritmos de deteção de anomalias, para criar perfis de comportamento normal de utilizadores e entidades. Desvios significativos a esses perfis são posteriormente registados, potencialmente correspondentes a ameaças.

O principal objetivo deste trabalho consiste em avaliar a aplicabilidade de uma *framework* de UEBA para proteger os clientes da Talkdesk, através da monitorização do *staff* dos *contact-centers* para detetar ameaças relacionadas com o roubo de dados sensíveis, ciber-fraude, abuso interno, entre outros. Começámos o trabalho pela exploração das fontes de dados mais apropriadas (para UEBA) disponíveis, e pela definição de cenários de ameaça, seguido da obtenção dos dados e implementação de pipelines de transformação, para converter dados os dados originais num formato adequado para os algoritmos de deteção de anomalias. De seguida, procedemos a uma análise exploratória dos dados, bem como seleção de *features*, e decidimos quais os algoritmos de deteção de anomalias a utilizar, bem como a estratégia de validação a implementar. Criámos um *dataset* artificial, através de conhecimento de domínio, e definimos parâmetros para variar (período de treino, *feature set*, contaminação), que aplicámos com 5 algoritmos distintos. Seguidamente, avaliámos cada combinação de parâmetros utilizando métricas de classificação supervisionada, com os algoritmos *autoencoder* e PCA a registar o *F1-score* mais alto: 0.97 e 0.95, respetivamente. Também avaliámos 3 métodos de interpretabilidade distintos, para explicar as anomalias reportadas.

Por fim, fizémos *deploy* da *framework* utilizando PCA e um mês de dados de 2 clientes, num ambiente de *staging*, registando um total de 76 anomalias causadas por 45 agentes distintos, com 67 verdadeiros positivos e 9 falsos positivos. Nós acreditamos que a *framework* está preparada para produção, requerendo apenas pequenos ajustes no algoritmo de interpretabilidade e uma estratégia para suprimir anomalias similares a outras anteriores, reportadas como falsos positivos através de *feedback* dos clientes.

Palavras-Chave

UEBA, Inteligência artificial, Machine learning, Cibersegurança, Deteção de anomalias, Deteção de fraude

This page is intentionally left blank.

Contents

Acronyms

cbLOF cluster-based local outlier factor. 10, 12

CVSS common vulnerability scoring system. 21

DIFFI depth-base feature importance for the isolation forest. 3, 17, 41, 58, 67

DLP data loss prevention. 1, 18, 20, 23

EDA exploratory data analysis. 3, 4, 31, 42, 53, 67, 68

ETL extract, transform, load. 43, 61

HBOS histogram-based outlier score. 3, 10, 37, 47–51, 53, 54, 67

IDS intrusion detection system. 1, 7, 17, 21

kNN k-nearest neighbor. 10, 11

LIME local interpretable model-agnostic explanations. 16

LOF local outlier factor. 10, 11, 21

ocSVM one-class support vector machine. 10, 12, 21, 23

PCA principal components analysis. v, 3, 10, 13, 14, 24, 32, 39, 40, 42, 47, 50, 52–55, 57, 59, 64, 65, 67, 68

PII personally identifiable information. 2, 20, 26, 31, 34, 44, 53, 64, 65

SHAP Shapley additive explanation. 3, 16, 17, 41, 42, 58, 67

SIEM security information event management. v, 1, 18, 21

SOC security operations center. 1, 19

SVM support vector machine. 12, 16

UEBA user and entity behavior analytics. v, 1–5, 7–9, 15, 17–22, 25–28, 30, 39, 43, 49, 56, 67, 68

XAI eXplainable artificial intelligence. 3, 16

This page is intentionally left blank.

List of Figures

This page is intentionally left blank.

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

Cybersecurity poses huge challenges in today's digital world, aggravated by the centralization of organizations' infrastructures in cloud environments and the growing trend for remote work accelerated by the pandemic of COVID-19.

Typical enterprise security operations centers (SOCs) tend to focus on external threats, by using rule-based monitoring tools such as security information event management (SIEM) along with several security products deployed over the network: firewalls, intrusion detection system (IDS), data loss prevention (DLP), and others. However, rule-based tools are becoming less and less effective, due to the emergence of new threat models that make perimeter-based security obsolete [1, 2]. According to the HIMSS's "2019 Cybersecurity Survey" [3], **the biggest threat actor groups** are "online scam artists", which include cases of phishing, spear phishing, business email compromise, responsible for 28% of all the security incidents, and **negligent insiders** (those who mean well but are negligent and facilitate or cause data breaches), which are responsible for 20% of all the incidents in 2019. Furthermore, 6% of the incidents are caused by **malicious insiders**, which on the contrary of "negligent insiders", intentionally steal information or compromise the infrastructure, and 11% of the threats are caused by hackers (*e.g.*, cybercriminals, bug bounty hunters).

With the traditional approaches, based on rules and thresholds, lacking the capability to detect novel threat scenarios, such as zero-day attacks, and fight insider threats, such as a compromised insider attempting lateral movement [4], novel strategies, based on artificial intelligence, are required to protect organizations, especially in domains with high exposure, such as contact centers.

1.1 UEBA for contact centers

In the last few years, artificial intelligence strategies in the cybersecurity domain are being adopted on a growing trend, with good results. One emerging approach to detect and prevent (mainly) insider threat scenarios is user and entity behavior analytics (UEBA), a cybersecurity process focused on defining baselines for user behavior. These baselines are then used to detect anomalies, indicating possible threats [5]. UEBA leverages machine learning in the form of statistical analysis and outlier detection to perform anomaly detection. With this approach, it is possible to detect security use cases such as **compromised users**, when users' credentials are stolen due to phishing strategies or use of weak passwords, **sensitive information theft**, when a user deliberately downloads/steals sensitive data from customers or the organization, **host compromise**, concerning entities/hosts being monitored to detect suspicious activity, **cyber fraud**, concerning financial data (treasury, payments), among others.

Founded in 2011, Talkdesk is a software engineering company, building an innovative cloud-based contact center product, with the mission of drastically improving customer experience. In 2020, Talkdesk has more than 1800 customers in 75 countries, which include IBM, Acxiom, 2U, Trivago, Canon, Glintt, and many others [6].

Talkdesk's contact center product is a critical point of contact between client companies and their customers. Contact centers are crucial channels that organizations have to acquire new customers, help customers with any issues they may have, know more about the way customers feel about them, and employ marketing strategies. Thus, the contact center and its agents have this major role in **improving customer experience**.

Moreover, there has been an increase in the popularity of cloud-based contact centers such as Talkdesk's. However, contact center jobs are usually considered to be **precarious** and with **high turnover rate** (agents tend to leave after a short period of time). With a high number of agents that do not see their job as a long-term career, **contact centers have an increased risk of suffering security breaches** in the form of the use cases mentioned above, such as agents granting their credentials to someone else, deliberately giving access to customers' personally identifiable information (PII).

According to "Cisco Contact Center Global Survey 2020" [7], where Cisco interviewed 700 contact center executives, 77% of the respondents agree that "Security and data privacy are key challenges in the contact center". Furthermore, Cisco describes contact centers as **"low hanging fruit for fraud"**, in the sense that a lot of customers' sensible information is accessible to agents that do not have a stable job.

Even though we are able to find and explore several successful general-purpose UEBA products in the market, such as Exabeam [8], IBM's QRadar [9] or Gurucul [10], **we did not find any application to the contact center domain**, which requires several specificities to monitor both the behavior concerning agents' call activity as well as actions performed in Talkdesk's application (*e.g.*, access to contacts' PII).

Considering all the reasons above, Talkdesk's product is a great opportunity for implementing a UEBA framework. The main objective of this work is thus to assess the applicability of such a framework in the contact center domain.

The framework proposed should be able to create baselines of the typical behavior of contact center staff (agents, supervisors, etc.), detect and alert on events that deviate significantly from those baselines, and provide an accessible interpretation for the causes of these events. To do so, we started by analysing two log sources of data considered to be relevant in the security and business perspective, that we expect to be also suitable for UEBA baselining: **call logs**, which contain information regarding each call, such as the agent to who the call was assigned, time metrics such as the waiting and talk time, and other data such as if the call was missed by the agent or abandoned by the customer, the call cost, etc.; and **audit logs**, which contain security-related information concerning the activity of users in Talkdesk's application, such as login events, accesses to sensitive data (customer PII) and call recordings, and others.

Using domain knowledge and in-depth analysis of our data sources, we defined example threat scenarios (combining typical UEBA use-cases with the contact center data used in this work) that we aim to be able to detect with the algorithms employed.

We proceeded with the implementation of data pipelines capable of aggregating the data over a determined time period, extract features and merge the sources into one single dataset. We then conducted a process of exploratory data analysis (EDA) on the features extracted, used to identify features that could be representative of user behavior, so that it is possible to use unsupervised anomaly detection models to create a behavior baseline and detect relevant deviations (from the security or business perspective). Several aggregation periods to model user behavior can be employed (*e.g.*, real-time detection with cumulative aggregation, 3-hour or 24-hour aggregation, and aggregate each user session - from login to logout), from which we choose to focus on users' daily activity (24 hours).

After finishing EDA, we used different libraries containing the anomaly detection algorithms we selected for this analysis, namely histogram-based outlier score (HBOS), the Mahalanobis method, isolation forest, principal components analysis (PCA) and a neural network algorithm, the autoencoder.

While explainability (or interpretability) of the detected anomalies is one of the requirements of the framework we aim to test, it is also one of the biggest challenges that emerge with more complex detection mechanisms. When an action or a group of actions is marked as an anomaly by any of the algorithms, it is of major importance to be able to explain the decision in a way that it is easy to understand without the need for technical knowledge of the data or the algorithms.

Interpretability of machine learning outputs is a known challenge in academia and in the industry, with the field of eXplainable artificial intelligence (XAI) providing several innovating approaches to overcome the interpretability issue associated with the more complex machine learning problems (*e.g.*, neural networks) [11], which creates distrust at various levels due to the inability of end-users to understand and easily interpret the outputs of the models. While XAI is making a lot of progress in the supervised domain, interpretability in unsupervised problems such as the anomaly detection algorithms typically employed in UEBA still lacks exploration. Nevertheless, we researched the different model-agnostic and model-specific interpretability approaches suitable for our problem and evaluated how three of them (z-score based, depth-base feature importance for the isolation forest (DIFFI) and Shapley additive explanation (SHAP)) perform in our problem.

Finally, we defined and applied an ambitious validation strategy, which consisted of several stages, namely:

- the creation of labeled examples of malicious behavior, to perform validation with classification metrics, such as AUC-ROC and F1-score;
- sampling behavior vectors from supervisors and administrators, as we expected a significantly different behavior when comparing to agents, using those as the positive class (and agents' vectors as the negative) for the same classification metrics, and assess whether the algorithm is capable of accurately distinguish between the behaviors of different roles;
- deployment in a staging environment, using data from 2 distinct clients with a different number of staff elements and working settings, in which we analysed the anomalies and tuned the system with a simulated feedback mechanism.

For this work, we were given access to a period of 3 months of real call and audit log events, required to ensure we begin this exploratory path using data that is representative and relevant to our goals. However, and even though all data studied in this work results of aggregations, the raw data contains sensitive information, a reason why we chose to exclude some of the visualization representations we used during EDA.

1.2 Objectives and contributions

As stated, the main objective of this work consists in the evaluation of the applicability of UEBA framework to deploy in Talkdesk's contact center environment. By modeling the contact centers' staff activity based on application and call logs, we should be able to detect abnormal deviations from the baselines defined, thus constituting possible security threats.

The main objective is subdivided into 4 fine-grained goals, namely:

- **Explore and select, between the data sources available, the most relevant indicators of potential threats**, suitable for behavioral profiling (in the contact center context) and anomaly detection. The log sources used and respective extracted fields are determinant in the success of the framework, reason why it is important to explore and understand how they can be used to model contact center users' behavior.
- **Select and compare different state-of-the-art anomaly detection algorithms**. There is a vast number of anomaly detection algorithms we can choose from, however, it is of utter importance to research and understand how the algorithms work, what assumptions they make (mainly regarding the data), whether they can scale up to the volume of data generated by Talkdesk's biggest clients and, ultimately, evaluate and compare multiple of them using our data.
- **Provide alerts (anomalies) that are easy to interpret by end-users**, namely the supervisors of the contact center. The ability to easily interpret an anomaly is a critical requirement in the cybersecurity domain, as it provides the insights necessary for analysts (that are not security experts) to clearly understand what triggered an alarm. Explainability in unsupervised anomaly detection problems is challenging but extremely necessary, reason why researching the techniques available and their applicability in our scenario is also a crucial objective.
- **Define and implement a validation strategy that ensures that the framework is applicable in a production environment.** All the objectives above require a strong validation strategy, that ensures the correctness of the results, a low false positive rate and that allows to detect improvement points. Validation in unsupervised machine learning is a familiar challenge in academia, but it is still possible to identify and apply different approaches capable of assessing the quality of the framework.

As for the main contributions, and to the best of our knowledge, this work is **the first application of a UEBA approach in the contact center domain**, one that is highly susceptible to security threats. We present an end-to-end assessment of the applicability of an ambitious framework, from data collection, transformation and analysis, to the comparison of several anomaly detection algorithms, as part of a heterogeneous strategy of validation.

Our work has the ambition of a production deployment, reason why we tackle important requirements such as the interpretability of the anomalies and scalability to a continuously growing volume of data. We also present the results of a deployment in a staging environment with real data, from two distinct contact center clients.

1.3 Thesis outline

The remaining of the document is divided into 5 chapters.

Chapter 2 contains the state-of-the-art analysis on anomaly detection and interpretability algorithms suitable for our problem. We also present a research on the UEBA solutions currently in the market, the advantages and disadvantages of UEBA when compared to the traditional cybersecurity solutions, as well as scientific articles on the topic.

Chapter 3 contains the analysis of the raw data collected for this work (3 months of real data), from different log sources, as well as the data transformation pipelines required to create a final dataset containing daily aggregated user behavior, followed by the exploratory analysis of that dataset.

Chapter 4 details the anomaly detection and interpretability algorithms selected to evaluate and compare. It also presents the characteristics of the final dataset used in the experiments.

Chapter 5 contains the implementation of the validation strategy defined, with the main discussion and results obtained. We present the experiments where we compare the algorithms selected, using synthetic anomalies created for that purpose, as well as an automated approach based on negative sampling. The chapter also contains the evaluation of the interpretability algorithms adopted and a mechanism used to collect feedback from the clients and monitor the quality of the models. Finally, a deployment in a staging environment is presented, using 1 month of real data from two distinct customers.

Chapter 6 details the conclusions and some of the plans of future activities to conduct.

Chapter 2

Background and related work

This chapter contains an introduction on the topic of anomaly detection, followed by an analysis of the state-of-the-art unsupervised anomaly detection algorithms, a subsection with an investigation on some explainability/interpretability methods, and finally an analysis of user and entity behavior analytics (UEBA), from typical security use-cases to products in the market and scientific approaches.

2.1 Anomaly detection

An anomaly is an event, set of events, or pattern that differs significantly from a predetermined baseline [12]. Anomalies can appear in data due to several reasons, such as malicious activity (credit card fraud, cyber-attacks, and others). If an event is anomalous regarding the baseline but has no interest for the analyst and the specific problem (*e.g.*, it could be introduced due to a measurement error) it is considered as **noise**. Since noise is typically an obstacle for data analysis, an approach for **noise removal** is often used before analysing the data. Figure 2.1 depicts example anomalies (marked in red).

Anomaly detection consists of building baselines/patterns of the normal behavior of a system, and detect any event that is anomalous when compared to that baseline [12, 13]. The topic is used extensively in fields such as credit card fraud detection, intrusion detection systems (IDSs) and fault detection.

This section focuses on concepts around anomalies, the main state-of-the-art challenges in

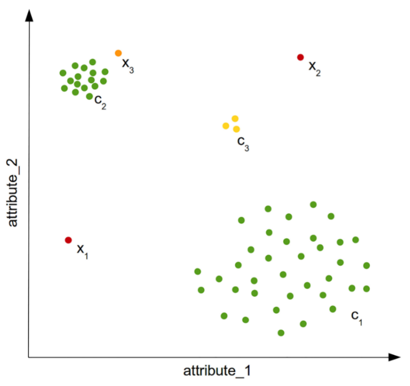


Figure 2.1: Example of anomalous points in a 2-dimension problem

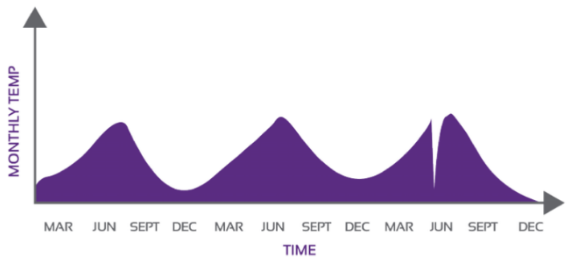


Figure 2.2: Example of a contextual anomaly.

anomaly detection, and possible validation strategies for unsupervised anomaly detection are also discussed.

2.1.1 Types of anomalies

Regarding the nature and properties of the data, an anomaly can be distinguished between a **point anomaly**, a **contextual anomaly** and a **collective anomaly** [12, 14]:

- Point anomalies: individual data instances that are anomalous with regard to the baseline, independently of the context. This is the most common approach in the anomaly detection domain.
- Contextual anomalies: when an anomaly is considered as such only in a specific context, based on the neighbors of the instance. A common contextual attribute is time, for instance in a time-series dataset. In the example of figure 2.2, there is an anomaly in June of the third year, and the temperature is similar to those seen in December of the first year. In June, there is a contextual anomaly, and time is seen as the contextual attribute.
- Collective anomalies: when a combination of many instances (points) is an anomaly, and the points by themselves are not. It typically occurs in datasets where instances are related, such as a sequence of logs in the security context.

Transforming contextual and collective anomaly problems into point anomaly problems

In security-related anomaly detection problems such as credit card fraud detection, the contextual attribute time of a transaction is relevant, as well as past transactions [15] and, in UEBA the same applies, as the typical approach is to detect anomalies consisting of a user's session. Furthermore, it may be relevant to determine if a user is interacting with the system in abnormal periods (*e.g.*, in the contact center domain, if the user is logging in out of the normal functioning hour of the contact center). Thus, it is necessary to transform contextual/collective anomaly problems into point anomaly problems, which is typically done with aggregation and discretization strategies [14]. In UEBA, users' logs can be aggregated with respect to several contexts, such as time (1 hour, the user session period, 24hours) the user's location, role within the organization, and others.

When it comes to the distance between the anomalies and other data points or clusters, anomalies can also be classified as **global anomalies** or **local anomalies** [13]:

- Global anomalies: In figure 2.1, the points x_1 and x_2 are clearly isolated and far from every other data point/cluster. Thus, they are considered global anomalies.
- Local anomalies: Still in figure 2.1, but now focusing on x_3, the point has some distance from the cluster C_2, but not too far as to be considered a global anomaly. Thus, x_3 may be regarded as a local anomaly.

In UEBA applications, both global and local anomalies can be relevant. While global anomalies may indicate a clear different behavior from the population, a local anomaly may indicate that specific users are changing their behavior with regard to their baseline or their cluster.

2.1.2 Supervised vs. unsupervised anomaly detection

Regarding the presence or absence of labeled anomalies/normal instances in the dataset, the problem of learning to detect anomalies can be classified as **supervised** or **unsupervised**.

In supervised learning, each data instance contains a label indicating whether it is an anomaly or not. Having access to such labels has advantages such as facilitating both the process of **feature engineering**, since the main goal of feature selection is to improve a pre-determined metric (such as ROC, precision, recall,...), and the **validation of the model**, since we can quantify how well the model works on training and test sets (bias/variance trade-off). On the other hand, there are two big disadvantages with supervised approaches: the skewness of the positive class, which means the portion of anomalies is far lower compared to the normal instances, which raises the so-called "imbalanced class issues" (addressed with strategies such as undersampling, as explored in [16]); and the challenge posed by obtaining useful and representative anomalies of all the different possible anomalous behaviors.

Unsupervised learning, on the contrary, does not require labels, and there is the assumption that normal instances are way more frequent than anomalies. In the UEBA problem we are trying to solve, due to the impossibility of acquiring labels representative of several anomalous behaviors, as well as the goal of detecting threats never seen before (which supervised approaches could fail to accomplish), **unsupervised learning will be used**.

2.2 Unsupervised anomaly detection

This section focuses on the main algorithms considered for the task of unsupervised anomaly detection suitable for **multivariate data**, which is the case of UEBA. We present these algorithms regarding how they work, the pros and cons, computational complexity regarding the need of scalability, which of them are suitable when it comes to explaining the outputs, and finally a summary comparison between them, which helps to decide which are the most appropriate for the problem of UEBA.

These algorithms work on two basic assumptions: anomalies are somehow different than regular events (either they are more distant to some reference point, or do not belong to any cluster, or become isolated with a decision boundary), and; the portion of anomaly events is much lower than the normal events in the data (*e.g.*, a ratio of 99% normal to 1% anomalies).

2.2.1 Anomaly detection algorithms

The following algorithms are detailed in the subsection below: histogram-based outlier score (HBOS), Mahalanobis method, k-nearest neighbor (kNN), local outlier factor (LOF), cluster-based local outlier factor (cbLOF) one-class support vector machine (ocSVM), principal components analysis (PCA), isolation forest and the autoencoder [12, 13, 14].

Histogram-based Outlier Score

histogram-based outlier score (HBOS) [17] is a statistical algorithm used in anomaly detection, and is characterized by its simplicity and scalability. HBOS is the only algorithm analysed in this work that assumes **feature independence**, this is, it applies a combination of univariate methods and does not model correlations between features.

The algorithm starts by computing a histogram for each feature. Different methods are applied depending on the nature of the data (for categorical data, simple counting is used; for numerical features, the algorithm allows either fixed or dynamic bin width), and the height of each bin corresponds to the **density estimation**.

After obtaining one histogram for each feature f, the histograms are normalized so that the maximum height is equal to 1.0, ensuring equal weight for all the features, and the anomaly score of instance x is calculated as shown in equation 2.1, where $hist_i$ corresponds to the height of the bin x corresponds to, in the histogram of feature i.

$$HBOS(x) = \sum_{i=0}^{f} log(\frac{1}{hist_i(x)}) \tag{2.1}$$

According to the authors, the complexity of HBOS is $O(n)$ if a fixed-bin width is used, and $O(nlog(n))$ in the case of dynamic bins.

Mahalanobis method

Statistical and probability models were the earliest methods for anomaly detection. Such models are typically used in detecting univariate extreme values (at the tails of the probability distribution of the data). Even though such methods have limitations in detecting multivariate and more "subtle" anomalies, there is one that stands out, which is the **Mahalanobis method**.

The Mahalanobis method [18, 19] assumes the (n x m) data to have a (multivariate) Gaussian distribution, and the distance between the points and the centroid of the data is calculated as follows:

$$Mahalanobis(X_i, \mu, \Sigma) = \sqrt{(X_i - \mu)\Sigma^{-1}(X_i - \mu)^T} \tag{2.2}$$

Where μ is the m-dimensional mean vector and Σ is the (m x m) covariance matrix. In the case that the covariance matrix Σ is not invertible, it is possible to compute a pseudo-inverse, using for instance singular value decomposition, as the authors do in [20]. After computing the inverse of the covariance matrix, the Mahalanobis distances of each point in the training set are calculated with equation 2.2, and the anomaly could be classified based on the statistical distribution of the training distances.

The main advantage of the Mahalanobis distance is the use of the inner-correlations of the data, which is ensured by the covariance matrix, allowing to overcome the main drawbacks of other extreme-value analysis statistical methods. Furthermore, the method is parameter-free, and is also effective in detecting outliers coming from extreme values. For the main drawbacks, the method assumes a specific data distribution (Gaussian), which could be a big limitation in real-world complex scenarios such as behavioral profiling, and requires a complexity of at least $O(m^2)$ to invert the covariance matrix, both in time and space.

K-nearest neighbors (kNN)

The kNN algorithm for anomaly detection is one of the most simple and straightforward, used to detect **global anomalies**. The procedure consists of finding the k closest records for each record in the dataset. After that, an anomaly score is calculated either by using the distance to the *kth* neighbor (referred to as kth-NN) or the average distance to all k neighbors (referred to as kNN) [13].

The parameter k should not be too low, as the density estimation (number of neighbors) may not be reliable, nor too large. The complexity of finding the k-nearest neighbors is $O(n^2)$.

Local outlier factor (LOF)

LOF [21] is a popular anomaly detection algorithm, and is capable of detecting both **local and global anomalies**. It uses kNN in the first step to obtain the k closest points to an instance (N_k), and uses the relative density for the detection of outliers, which makes the algorithm capable of detecting local anomalies.

The reachability distance, RD_k, between two points, x and *x'*, is given by equation 2.3, where D_k represents the distance to the *k-th* nearest neighbor and D is a distance such as the Euclidean distance.

$$RD_k(x, x') = max(D_k(x'), D(x, x')) \tag{2.3}$$

The inverse of the average reachability distance from the point x from its neighbors is given by the local reachability density (LRD_k), calculated using equation 2.4, where $|N_k(x)|$ denotes the total number of points present in the neighboorhood of x (which is equal or higher than $N_k(x)$, if there are multiple points at the same distance as the *k-th* neighbor of x).

$$LRD_k(x) = \frac{1}{\sum\limits_{x' \in N_k(x)} \frac{RD(x, x')}{|N_k(x)|}} \tag{2.4}$$

Finally, the LOF score is calculated with equation 2.5, which uses a ratio between the average density of x's local density. The score of normal instances takes values of 1.0 or less, while outliers take larger values (higher than 1.0) as the local density of x lowers.

$$LOF_k(x) = \frac{\sum\limits_{x' \in N_k(x)} \frac{LRD_k(x')}{LRD_k(x)}}{|N_k(x)|} \tag{2.5}$$

Regarding complexity, the cost of the first step is $O(n^2)$, and the cost of the remaining 2 steps can be neglected.

Cluster-based local outlier factor (cbLOF)

CbLOF [22] uses clustering as a strategy to determine the most dense areas (clusters) in the data and performing density estimation for each cluster afterwards. It assumes that outliers appear in small sparse clusters or on the peripheral of a cluster [23]. Even though any clustering algorithm can be used for the first step, *k-means* is the most common choice, due to its popularity along with having low computational complexity.

After the first step, cbLOF classifies the clusters according to their dimension, and an anomaly score is computed based on the distance of each instance to its cluster center multiplied by the instances belonging to its cluster [13].

Similarly to k-nearest neighbors, the predefined number of clusters is critical for the success of the algorithm, and several experiences must be made.

CbLOF's computation cost is linear [22] ($O(n)$), faster than the distance-based approaches covered, especially when *k-means* is used as the clustering algorithm.

A variation of cbLOF, proposed as Unweighted-CBLOF or **u-CBLOF**, mitigates the effect of too small or large clusters in cbLOF. In this approach, the number of instances in the cluster is not used.

One-class support vector machine (ocSVM)

OcSVM is an extension of support vector machine (SVM) typically used in semi-supervised problems, due to its sensitiveness to the presence of outliers in the dataset. However, the approach can also be used in unsupervised problems.

When used for a classification task in supervised learning, SVM attempts to separate two classes (positive and negative) using a decision boundary. However, this one-class variation is named as such due to the assumed existence of only one class (normal or non-anomalous instances). Therefore, the boundary in this case (a hyper-plane or hyper-sphere) attempts to separate the data points as much as possible from the origin, after a transformation of the input points into a high-dimensional space, using a **kernel** [24].

Kernels are functions that map the input space into a high-dimensional feature space, where the data is more easily separable. Kernels can be of several types, which include linear, polynomial, sigmoid, or one of the most popular, the radial-basis function (RBF).

OcSVM also leverages the soft-margin approach, which allows the model to misclassify some examples to keep the margin as wide as possible, making the algorithm capable of accounting for outliers in the training dataset [23].

Even though it is hard to estimate the computation complexity of ocSVM, according to [23], it can range between a little less than quadratic to quadratic ($O(n^2)$), depending for example on the kernel chosen.

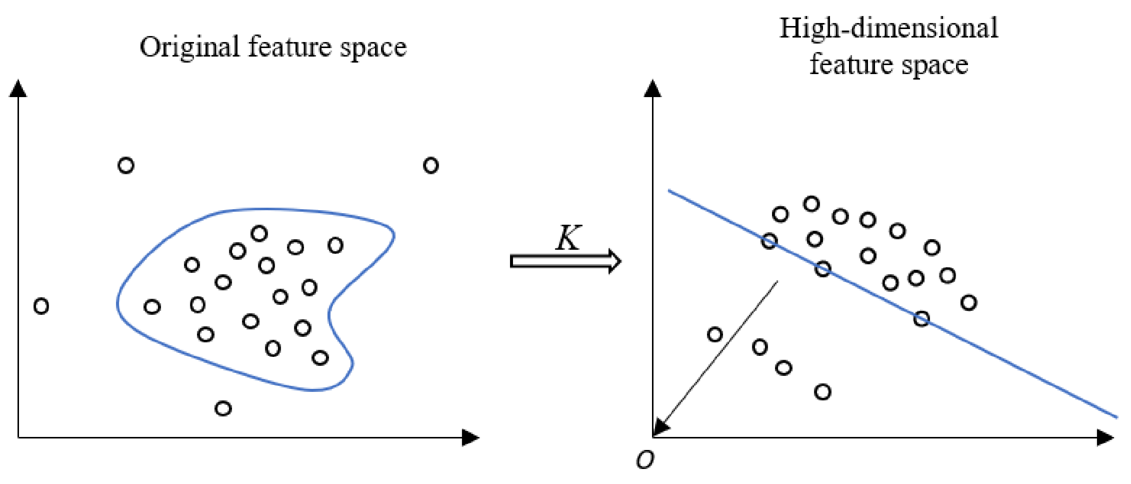


Figure 2.3: Example projection from the original feature space to a high-dimensional feature space where data is easily separable from outliers, using a kernel K.

Principal component analysis (PCA)

PCA is a common algorithm for detecting subspaces in the data, a technique known as dimensionality reduction, and is also suitable for anomaly detection, based on the assumptions that instances that deviate from the derived subspaces may indicate anomalies. PCA has the limitation of assuming that the data is highly correlated and aligned in lower-dimensional subspaces [19], which is most properly not verified in several real-world scenarios. The algorithm consists of the following steps:

1. Data standardization: a critical preprocessing step for PCA consists of standardizing the data (calculated by subtracting to the instance's the population mean and divide it by the population standard deviation), to ensure all the variables are in the same range;

2. Calculate the covariance matrix, which contains the covariance between each pair of variables, with the variance of each variable in the diagonal;

3. Calculate the principal components (also known as *eigenvectors*) of the covariance matrix, which represent the directions of the maximum variance in the data, done in an iterative fashion: the first component represents the direction of the maximum variance, the second component represents the second-highest variance direction, and so on. These components are orthogonal and thus uncorrelated;

4. Choose the number of principal components to preserve (the d first components that preserve the most variance), and discard the remaining (for example, a typical threshold would be: keep the d components that retain 90% of the variance);

5. Project the original standardized dataset into the new subspace given by the components, done by multiplying the original dataset by the matrix of the principal components.

After applying these steps, the points that lie further away from the hyperplane given by the principal components are classified as outliers.

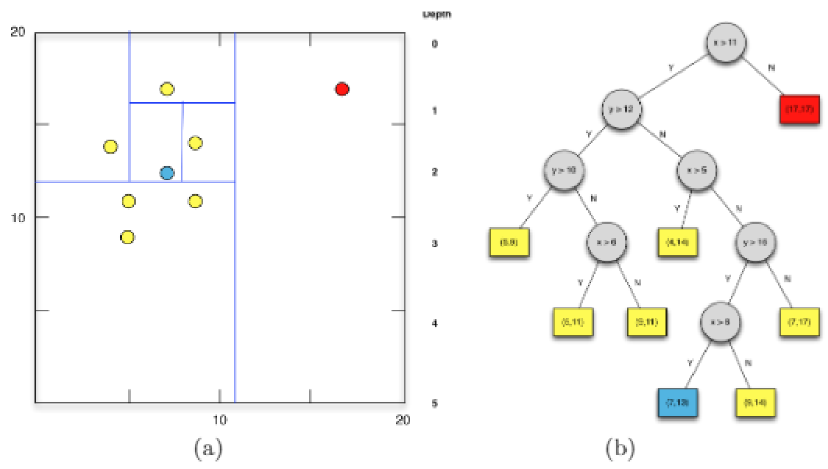


Figure 2.4: Example of a binary tree partitioning in isolation forest, from [28]. In the example, the point marked in red is isolated in just one iteration, while the point in blue (the mediod) takes 5 iterations to isolate.

Instead of explicitly removing the principal components (eigenvectors) that retain the least of the variance, as explained in step 4. (referred to sometimes as hard-PCA), we could use weighted distances across all the eigenvectors. This soft approach would be similar to the **Mahalanobis method**. The anomaly score for each instance is computed by calculating a squared sum of the normalized distance of each point to the centroid of the data along the direction of each eigenvector, weighted by the eigenvalue of that eigenvector [19].

The principal components (eigenvectors of the covariance matrix) are highly affected by outliers, which may lead to a poor estimation of the most appropriate directions to project the data. To solve this issue, a robust-PCA version (such as the one in [25]) could be used, in which the original data matrix M is decomposed into a low-rank matrix L_0 (the true values) and a sparse matrix S_0, as equation 2.6 shows.

$$M = L_0 + S_0 \tag{2.6}$$

To achieve this decomposition, and recover the low-rank matrix L_0, techniques such as principal components pursuit [25] or the augmented Lagrange multiplier method [26] are used.

The computation complexity of PCA is typically linear regarding the number of rows in the dataset, but quadratic regarding the number of dimensions $O(n + d^2)$ [12].

Ensemble: isolation forest

Isolation forest [27, 28] is an algorithm designed for unsupervised anomaly detection, based on the assumption that by representing the data as a binary search tree, anomalies are more susceptible to isolation, meaning being inserted as leaves at a lesser depth than normal instances. An example of this assumption is present in figure 2.4.

In isolation forest, n samples of a dataset D with N dimensions are used to build an ensemble of n trees. Each tree is built by selecting a random dimension x_i with $i \in \{1, 2, ..., N\}$, selecting a random value v within the minimum and maximum values that dimension takes. All the instances with a value smaller than v for x_i go through the left branch, while instances with higher values than v go through the right. This is done

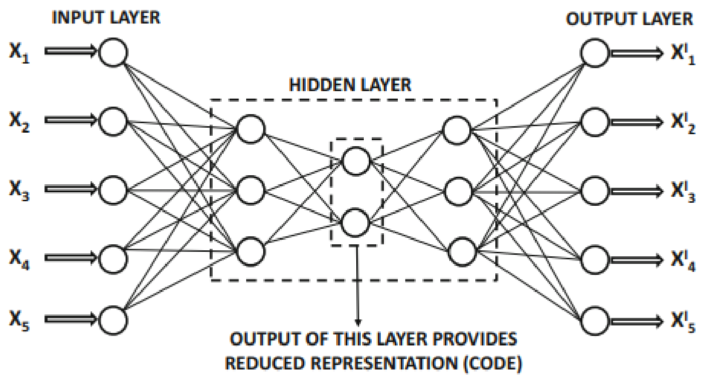


Figure 2.5: Autoencoder architecture (from [19])

recursively until a tree like the one in Figure 2.4 is built or a limit depth is reached.

After several trees are built with the steps above, the evaluation of whether a new instance is an anomaly or not consists of running it through all the trees and averaging the insertion depth for that instance.

The authors in [29] propose an "Extended Isolation Forest" approach, which uses a random slope for branch cut, and not horizontal or vertical cuts only, as seen in figure 2.4.

Isolation forest has a linear computational complexity [27] ($O(n)$), which makes it a great candidate for the UEBA problem at hand.

Autoencoder

Autoencoders are deep neural networks that use a bottleneck architecture to create a representation of the data with lower dimensionality, and then attempt to reconstruct the input.

Figure 2.5 shows an example of an autoencoder with 3 hidden layers and an output layer with the same dimension as the input layer. The goal is to train the network to minimize the aggregated reconstruction error given by $\sum_{i=1}^{n}(x_i - x'_i)^2$, where x'_i is the reconstruction of x_i. The main assumption is that outliers have higher reconstruction errors, while normal instances have errors closer to 0.

The architecture of the autoencoder is typically symmetric on both sides of the middle layer (code), thus being possible to divide the autoencoder into two parts, the encoder and the decoder.

The autoencoder is also suitable for dimensional reduction but, when compared with PCA, the first does not assume a linear compression, thus being able to apply a nonlinear reduction of the dimensionality, which makes the autoencoder a more powerful approach. However, **drawbacks** such as the difficulty in effective training (avoiding overfitting, complex parameterization), the training time (neural networks are slow to train, even though it is difficult to estimate a time complexity), and the sensitivity to outliers in the training data could make it difficult to apply such architecture on early stages of the UEBA framework.

2.2.2 Anomaly explainability/interpretability

One of the most important challenges/difficulties in applying anomaly detection in the cybersecurity field is the ability to explain the anomalies reported, especially with complex models such as SVM, neural networks, or ensemble methods (isolation forest), where it can be difficult to associate an output of the model to the features most important in the decision. EXplainable artificial intelligence (XAI) is a field with the goal of providing solutions for the interpretability problem: the problem of producing AI results that can be interpreted easily by humans.

The most straightforward way of ensuring explainability of the results would be to use models that are interpretable by nature, which include linear regression, logistic regression, and decision trees. However, such models are used in supervised learning tasks. For the models covered in section 2.2, which are, all of them, not interpretable by nature, more advanced techniques are required.

It is possible to find vast bibliography on explainability for supervised models, but the same does not happen for unsupervised problems such as the anomaly detection domain. However, some relevant approaches exist, mainly model-agnostic (such as local interpretable model-agnostic explanations (LIME), Shapley additive explanation (SHAP)), but also a model-specific approach proposed recently, DIFFI for isolation forest.

LIME and SHAP

LIME [30] and SHAP [31] are two popular model-agnostic approaches to AI explainability.

LIME consists of using an explainable linear model g (even though other explainable models could be possible, such as decision tree or rule list), a measure of complexity of the explanation provided by g, $\Omega(g)$ and a measure of how unfaithful g is in approximating the model being explained, f, in the locality defined by π_x, given by $\mathcal{L}(f, g, \pi_x)$.

The goal of LIME is to minimize the loss, given by $\mathcal{L}(f, g, \pi_x) + \Omega(g)$, to ensure both local fidelity and interpretability. Sampling is used to learn the local behavior of f, by extracting instances around the one being explained (x), weighted by π_x (higher weights for instances closer to x). The authors validate LIME by using it for selecting between different classifiers (explainability may prove a classifier with higher accuracy to be worst), providing trustworthiness for the predictions and the model, detecting leakage in the model features, and feature engineering tasks.

Similar to LIME, SHAP also uses local explainability, along with cooperative game theory concepts. SHAP values are used as a measure of feature importance, based on the Shapley values of a conditional expectation function of the original model. The SHAP values for a feature represent the change in the expected prediction when conditioning on that feature, and explain how to get from the average model prediction if no features were known to the actual prediction.

Even though model-agnostic techniques have the great advantage of being highly portable throughout different models, they also contain limitations such as the **increased computational costs** and not leveraging the internal structure of the algorithms. Model-specific approaches, on the other hand, are able to circumvent these disadvantages.

DIFFI for isolation forest

Even though we were not able to find a significant number of model-specific interpretability methods for anomaly detection, we did find a recent approach called depth-base feature importance for the isolation forest (DIFFI) [32] which stands out. The authors present DIFFI as a global interpretability method, which provides global feature importance for the isolation forest, but also include a local version of DIFFI, for interpretation of individual predictions and a procedure to perform unsupervised feature selection.

DIFFI starts with two assumptions to consider a feature as 'important' to the anomaly detection task: a split associated with that feature should both induce the isolation of anomalous instances at small depths and produce higher imbalance on anomalous instances while being useless on regular points. The approach consists in computing cumulative feature importances (CFIs), real values for both inliers and outliers that are combined to produce the final feature importance measures, obtained based on the assumptions above.

Regarding the experimental results, the authors of DIFFI use synthetic and real-world data to evaluate the local interpretability model and the feature selection method. They conclude that local-DIFFI performs as effectively as the upper-mentioned SHAP, at a much lower computational cost, making it more suitable for applications in production environments.

2.2.3 Algorithm comparison and summary

Table 2.1 contains a summary comparison of the algorithms described above. It summarizes the most important requirements for the UEBA framework, which consist of the time complexity of the model (the time it takes to train), the robustness to high dimensionality (an increase of the number of features/dimensions), whether it is possible to interpret/explain the anomalies reported, and the python libraries that have an implementation of each algorithm.

By observing the table, the isolation forest is probably the algorithm that stands out the most, due to a linear time complexity, by being the only one with an interpretability method available, being robust to high dimensionality, and with availability in the libraries scikit-learn and pyOD. However, this does not mean that other algorithms should not be applied and compared, as these advantages are not enough to predict if the isolation forest will have better results than the other approaches.

2.3 UEBA - machine learning for cybersecurity

As stated in [33], nowadays it is "*impossible to deploy effective cybersecurity technology without relying heavily on machine learning*". In fact, domains such as spam email classification, IDSs, credit card fraud detection, all rely on machine learning to detect security threats with a growing effectiveness [34]. While in spam classification, typically, supervised classification is used to predict if a new coming email is spam or ham, based on millions of previously labeled emails, IDSs and fraud detection often use **unsupervised anomaly detection** to detect deviations from a typical behavior: the former looks at the behavior in the network, and attempts to detect suspicious traffic, and the latter tries to identify suspicious credit card transactions based on past transactions.

UEBA is a cybersecurity process focused on building profiles (baselines) of user and entity

Table 2.1: Algorithm comparison

Algorithm	Time complexity	Explainability algorithms	Robustness to dimensionality	Other limitations/observations	Python implementations
HBOS	$O(n)$	Not needed	Yes	Assumes feature independence	pyOD
Mahalanobis method	$O(d^2)$	-	Yes	Assumes underlying data distribution.	scikit-learn
kNN	$O(n^2)$	-	No	-	pyOD
LOF	$O(n^2)$	-	No	-	pyOD
cbLOF	$O(n)$	-	No	-	pyOD
ocSVM	$O(n^2)$*	-	Yes	Difficult parametrization; * hard to estimate the complexity due to the parametrization options	scikit-learn pyOD
PCA	$O(n + d^2)$	-	Yes	Assumes the data is highly correlated and aligns in a lower dimension subspace; Difficult to interpret the outputs.	pyOD
Isolation forest	$O(n)$	DIFFI	Yes	-	scikit-learn pyOD
Autoencoder	-	-	Yes	Slow to train (high time complexity, even though it is hard to estimate); Hard to train (high parametrization); Difficult to interpret the outputs.	pyOD

Table 2.2: UEBA vs. SIEM, adapted from [35]

UEBA	SIEM
Threat detection based on machine learning	Threat detection based on a set of rules
Capable of detecting novel threats	Not capable of detecting novel threats, due to the limitations imposed by the rule definition
A better option for insider threat detection	A better option for external threat detection
Generates risk scoring for users and entities	Generates alerts on security events
Automated threat detection, thanks to machine learning	Needs a manual analysis on the reported information

(hosts, applications, endpoints, data storage) behaviors throughout time, and detect significant deviations, either from the user or the peers' baseline. UEBA systems typically collect data from several sources (raw logs, security information event management (SIEM) data, data loss prevention (DLP) data, network traffic packets) and leverage anomaly detection machine learning models to create alarms/visualization for suspicious events [4, 35].

As stated in [35], it is important to distinguish between UEBA and SIEM when it comes to the capabilities of each of them. SIEM consists of a complex set of tools and processes with the goal of giving a complete view over an organization's security. It monitors and aggregates events from firewalls, OS logs, network traffic logs, etc., and uses a threat detection model based on rules inserted by security practitioners to create alerts representing potential threats. Since several organizations adopt SIEM solutions for their security strategy, it may not be clear how advantageous a UEBA solution may be to complement it. Table 2.2 contains the main differences between UEBA and SIEM when it comes to their functionality.

UEBA has important advantages, that make its adoption more and more relevant for organizations exposed to threats posed by insiders:

- Capability of detecting unknown/never seen threats (novelty detection): with the unsupervised learning strategy defined, the anomalies reported consist of significant **deviations of any type** from a baseline; in contrast, either rule-based approaches such as those widely seen in SIEM tools, and supervised approaches, are designed to

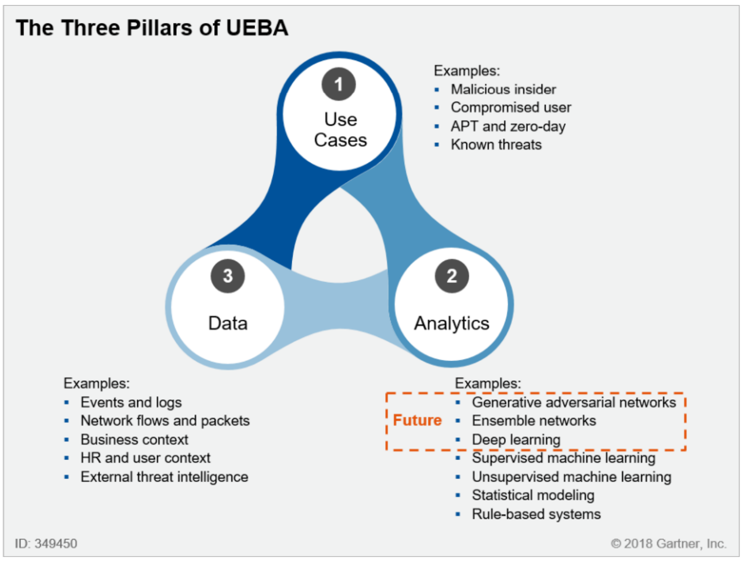


Figure 2.6: The three pillars of UEBA (from [4])

detect threats seen previously, due to the intrinsic formulation of the approaches;

- Appropriate for detecting both internal and external threats. Rule-based systems tend to fail for internal threat detection, as malicious insiders' activity is typically less explicit than external attacks, reason why it is harder to express insider threats with rules. On the other hand, by analysing and monitoring the users' behavior, it is possible to detect meaningful and potentially malicious deviations;

- Provide risk-based scoring/analysis over users and entities, based on how much the behaviors deviate from their baseline. This allows to quantify how likely a user is to incur in new threats.

On the other hand, some disadvantages include the **possibility of a high false positive rate**, especially when there is not enough data to train the models and create baselines, or when specific jobs are not easy to baseline. UEBA is not a replacement for other cybersecurity controls (such as firewalls, anti-virus, security operations center (SOC), etc.) and it is a complex system to adopt and deploy, probably making it less suitable to smaller businesses.

Figure 2.6 contains the "Three Pillars of UEBA", as presented in Gartner's 2018 "Market Guide for User and Entity Behavior Analytics" [4]. The pillar corresponding to "Data" represents several sources of data, which is ingested and processed in feature vectors to feed Machine Learning models.

The models, part of the "Analytics" pillar, consist of supervised or unsupervised machine learning, depending on the availability of labels, or even simpler models such as statistical or rule-based systems. In the future, it is expected that neural network-based models replace the current models.

The use cases pillar is approached in detail in the next subsection.

2.3.1 Typical use-cases and threat scenarios

As mentioned in the previous section, the first pillar of UEBA, as described by Gartner, is the use cases such systems can identify. Exabeam [36] and Gurucul [37], both vendors of UEBA products, provide a list of use cases, which are described in detail in this subsection. With this analysis, we have the goal of being able to identify some threat scenarios we think may happen in the context of the UEBA framework for the contact center environment, and which will hopefully explain some of the anomalies reported.

Account compromise/hijacking

In this use case, it is assumed that an attacker gains access to the credentials of a user, by exploiting vulnerabilities such as remote code execution or brute force. The UEBA machine learning models can use features such as location, IP address, device, and network packets to detect such anomalies. This allows a more advanced detection than the one performed by rule/signature-based systems.

High Privilege Abuse

This use case is related to irregular actions on accounts with high privileges. The UEBA framework ingests account data (including permissions), from identity and access management systems, directory services (such as Active Directory), and monitors those accounts to detect anomalous actions such as assigning high privileges to other accounts, access to classified/sensitive information, different IP addresses or locations. The detection of such threat scenarios is crucial to prevent attackers from gaining access to sensitive information and initiate other attack avenues.

Data exfiltration

This use case is related to anomalous extractions of sensitive data (such as personally identifiable information (PII)). The UEBA framework ingests data from DLP or other important data storage locations. The Machine Learning models can detect irregular extraction of sensitive documents, copying these to USB drives, as well as email content, source code, and others. By creating baselines of access to sensitive data, which include the normal quantity of data extractions, it is possible to detect anomalies with regard to those parameters.

Insider threats

Thanks to the capability of creating user behavior baselines and compare users with peers, UEBA is capable of detecting malicious insiders, assuming their actions are different from the peers. The framework is capable of assigning risk scores to users based on how much they deviate from peers and provides important insights through dashboards and alarms.

Cyber fraud

UEBA systems are capable of detecting cases of fraud in financial areas (treasury, accounting, payments). The system should ingest and monitor data concerning payments and other financial transactions, and detect anomalies. It is thus possible for an organization to integrate its fraud models with user and entity analysis.

Trusted Host compromise

Not only is it crucial for organizations to monitor the users, but it is also as important to monitor hosts and entities connected to the network, since if compromised, they are a source of access for attackers. To do so, UEBA integrates logs/data from security alerts, vulnerability scan results, common vulnerability scoring system (CVSS) and others, to detect anomalies concerning activity in hosts/entities.

2.3.2 Existent UEBA products and applications

Even though UEBA is a recent approach in cybersecurity, there are some vendors that offer UEBA products, either integrated with SIEM or as standalone offers. Example of such vendors are Gurucul [2], Varonis [35] and IBM [9].

In Gurucul's UEBA data sheet [38], they describe the process of real-time user risk scoring, which starts with the ingestion of data from several sources (security data from firewalls, IDSs, anti-virus software, infrastructure logs from servers, DNS, network logs from Netflow and Packet Capture, application audit logs, device attributes and details), processed and fed to machine learning models to identify anomalous behavior. Even though Gurucul does not disclose the machine algorithms used, they indicate some appropriate algorithms for unsupervised learning in [39], which include DBSCAN, LOF and ocSVM.

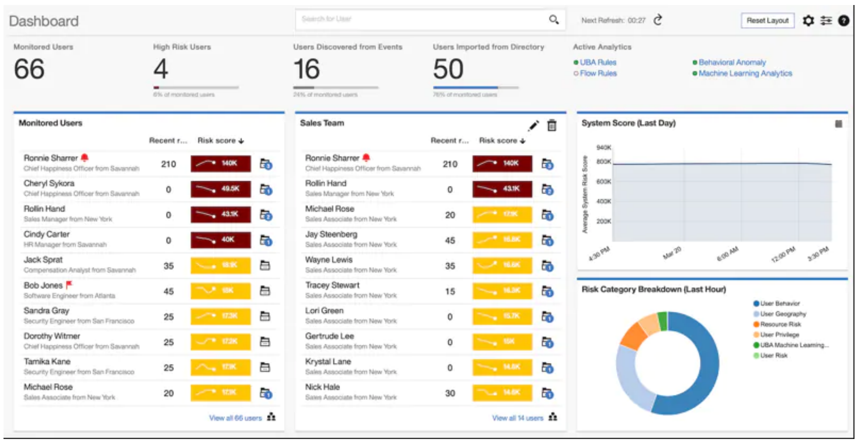
The capabilities of the product include: **predicting and preventing both known and unknown threats** by identifying activities deviating from baseline behaviors established by several machine learning models; **take action on alerts based on the risk severity**, to allow security analysts to prioritize the issues; **supporting several sources of big data**, to enable data ingestion from the customers' data lakes; **automating risk-based response**, through automation of the decisions based on the risk scores generated, such as increasing security controls for high-risk users.

IBM's QRadar is a popular SIEM that offers UEBA capabilities. The key features announced in [9] are the ability to detect insider threats based on behavioral anomalies, the integration with the SIEM product, and the generation of risk scores for each user under monitoring. Figure 2.7 shows the main dashboard for UEBA, which includes the monitored users along with a risk score, the risk categories, and alerts for new events.

Aside from released and available products, other relevant contributions in academia on UEBA exist.

The authors in [40] attempt to detect anomalous user behavior in mobile wireless networks, by using k-means clustering and hierarchical clustering to model call and SMS activity of users and detect anomalies. They use the aggregation of events into one-hour intervals to detect contextual and collective anomalies, by summing call and SMS activity for each user. Since this was an unsupervised approach, no objective results are presented, but the authors claimed to have compared the anomalous events with ground truth information, and that they correspond in fact to highly populated events, thus confirming high network traffic demand.

In [20], the authors present an overview of the Niara (Cybersecurity firm bought by HPE in 2017 [41]) UEBA modules. In the paper, they attempt to perform anomaly detection



Dashboard

Search for User

Next Refresh: 00:27

Reset Layout

Monitored Users	High Risk Users	Users Discovered from Events	Users Imported from Directory
66	4	16	50
	6% of monitored users	24% of monitored users	76% of monitored users

Active Analytics

- ● UBA Rules
- ○ Flow Rules
- ● Behavioral Anomaly
- ● Machine Learning Analytics

Monitored Users

	Recent r...	Risk score ↓
Ronnie Sharrer Chief Happiness Officer from Savannah	210	140K
Cheryl Sykora Chief Happiness Officer from Savannah	0	49.5K
Rollin Hand Sales Manager from New York	0	43.1K
Cindy Carter HR Manager from Savannah	0	40K
Jack Sprat Compensation Analyst from Savannah	35	18.1K
Bob Jones Software Engineer from Atlanta	45	18K
Sandra Gray Security Engineer from San Francisco	25	17.3K
Dorothy Witmer Chief Happiness Officer from Savannah	25	17.2K
Tamika Kane Security Engineer from San Francisco	25	17.1K
Michael Rose Sales Associate from New York	20	17.0K

View all 66 users

Sales Team

	Recent r...	Risk score ↓
Ronnie Sharrer Chief Happiness Officer from Savannah	210	140K
Rollin Hand Sales Manager from New York	0	43.1K
Michael Rose Sales Associate from New York	20	17.0K
Jay Steenberg Sales Associate from New York	45	16.8K
Wayne Lewis Sales Associate from New York	35	16.6K
Tracey Stewart Sales Associate from New York	15	16.3K
Lori Green Sales Associate from New York	0	15.7K
Gertrude Lee Sales Associate from New York	0	15K
Krystal Lane Sales Associate from New York	0	14.8K
Nick Hale Sales Associate from New York	30	14.6K

View all 14 users

Figure 2.7: IBM QRadar's UEBA dashboard (from [9])

on users based on the access and interaction with a specific server. To do so, they create two distinct baselines: an **historical baseline**, which only uses data from each specific user, to detect deviations from their own behavior; and a **peer baseline**, which allows the comparison of users with their peers (other users that behave similarly). To define the behavior baselines, the authors use the following daily aggregated features:

- timestamp of the first access of the day;
- timestamp of the last access of the day;
- duration between last and first access;
- sum of the duration of all eflows of the day;
- number of eflows during the day;
- total download bytes;
- total upload bytes of the day.

The authors use an enhanced version of the **Mahalanobis distance**, featuring one-sided deviations, which consists in ignoring deviations from the positive or negative side of the mean (*e.g.*, the download data below the mean may not be too relevant in a security perspective, while high quantities of download are highly concerning); **variable weighting**, which allows customers to give more relevance to determined features they are more concerned with, either in a business or security purpose; **robustness** to outliers, leveraging an implementation of **robust PCA** (rPCA); **explainable results**, by calculating the contribution of each feature to the anomaly; **learning from user feedback**, allowing to improve the models continuously.

Regarding Niara's UEBA workflow, the authors divide it into four steps: **data preparation**, where data is ingested from the available sources and grouped by entities; **feature extraction**, in which the relevant fields for UEBA are extracted and grouped by user/entity and aggregated per day; **behavior profiling**, corresponding to applying the Mahalanobis method to generate behavior profiles for each user; and finally **anomaly detection**, where the test values are scored against the profiles, generating anomalies and alerts. The **validation** approach presented is empirical, as the system is deployed in a real-world scenario, and anomalies are evaluated manually to assess the correctness of the results.

In [42], the authors propose an ensemble approach of three unsupervised algorithms: **isolation forest**, **ocSVM** and a neural network model similar to an **autoencoder** to detect anomalous user behavior. They collect data from system logs, application logs (such as DLP logs), user directory logs and others, from a software company and for a period of 3 months, consisting of normal behavior of 4 different users. Afterwards, the authors are able to simulate and label anomalies. The feature extraction steps consist in 24-hour aggregation of a total of 24 features, which include:

- total number of connections;
- timestamp of the first login;
- timestamp of the last login;
- total download bytes;
- total upload bytes;
- number of mkdir success;
- number of mkdir fail;
- number of delete success;
- number of delete fail.

Other features include mostly the success and failure of several operations (rename, download, upload, rmdir,...). These feature vectors are fed to each of the three models, and an ensemble approach with strictly filtering is applied to get the final result.

The authors have access to labeled data, and thus explore different scenarios when it comes to testing and validation: they train with different contamination rates (0%, 2.06%, and 4.03%). They conclude that **ocSVM is the best performer** with an anomaly-free dataset, but suffers a lot when the training data contains outliers (the recall and accuracy drop from 100% and 96.72% to 75.86% and 81.39%, respectively, when the dataset contains a 4.03% percentage of contamination). A similar scenario is observed with the neural network algorithm, with a drop from 95.62% and 97.70% to 79.56% and 72.41%, in the same conditions. In this study, **isolation forest is the model that performs the worst**, with an accuracy below 80% independently of the contamination rate in the training data.

In [43], the authors (from Exabeam) represent a user's **daily** traffic (collected from multiple sources, such as active directory, VPN logs, proxies) as an **array of event counters**. A table with more than 60 events is present in the paper, which includes:

- account-related events (account creation, deletion, lockout, password change or reset);
- authentication events (failed, successful);
- database events (database alert, login, query);
- DLP events (dlp alerts, dlp email alerts);
- operations on files (file delete, file permission change, file read, file write);
- privileged access operations;

- remote access operations;
- security alerts (from other security products);
- usb activity;
- and many others.

As explained, an array with a count for each of the events is stored as a user's activity for each day. The authors weigh each event by giving more importance to events that do not occur often (such as password changes), and use PCA to model user behavior and detect anomalies. A **hard-PCA** (presented in subsection 2.2.1) is used, meaning only the top eigenvalues are retained, bounded by a threshold *h*.

Chapter 3

Data analysis and threat scenarios

This chapter contains the steps conducted with the goal of obtaining, from the raw data which we were given access, a final dataset of grouped behavior activity, to feed the anomaly detection algorithms, and is summarized by the following steps/sections:

- **Analysis of the data available** for this work, in which we present the raw data collected;
- Definition of some example **threat scenarios** that we should be capable of detecting with the data selected in the first step;
- Implementation of the **data transformation pipelines**, in which data is aggregated to represent the behavior of users in determined periods (*e.g.*, 24h);
- Exploratory data analysis, in which we obtain insights and attempt to select the most appropriate features for behavioral profiling, from the whole set of features extracted from the original data.

3.1 Analysis of the data sources available

A successful user and entity behavior analytics (UEBA) implementation is characterized by the usage of rich and varied sources of data, containing several behavior indicators suitable for profiling/baselining. Therefore, it is of utter importance to start with data from which we are able to retrieve several metrics characterizing different patterns of interaction between the user and the contact center application. At this point, we have access to three distinct data sources: **call history** data, which contains information regarding every call, with several metrics presented below; **audit logs**, in which several different activities performed by users (create a session, change password, read a recording, etc.) are registered; and **roles**, which contains the role an agent is assigned (agent, supervisor, administrator).

Call logs

From the call logs, we extract the fields in table 3.1 (the field name does not correspond to the actual name in the logs). Each event (row in the data) corresponds to a call.

With the call data, we intend to model and detect anomalies regarding changes in the call behavior, such as when talk/wait times or the number of missed calls change abruptly,

Table 3.1: Fields from calls historical data

Field	Description
Call start time	Timestamp of the start time of a call
Call finish time	Timestamp of the finish time of a call
Call direction	"inbound" for calls received and "outbound" for calls sent)
Call missed	Takes the value "True" if the call was missed by the agent
Call duration	Total duration of the call
Call talk time	Total time a customer is effectively talking to an agent
Call out of working hours	True if the time of the call is outside the working hours of the contact center
Call cost	Cost of the call, in unknown unit/currency
Call waiting time	Time that the customer waits for the call to be answered

Table 3.2: Audit log operations used in this work.

Operation	Description
login_submit	User submits a login form, from any of the possible platforms.
create_session	A session is created (or renewed) for a user.
remove_session	A session is revoked for a user.
change_password	A (logged-in) user changes the password.
reset_password	A (not logged-in) user resets the password.
read_contact	An access to a contact's PII.
read_recording	An access to the recording file(s) of a call.

possibly indicating a decrease in performance or someone else using the application, and calls in unexpected/inappropriate hours, such as the period when the contact center is closed.

Audit logs

Audit logs consist of events corresponding to the activity of users. Table 3.2 contains some of the audit events implemented in Talkdesk and used in this work (the operation names were modified). Each event corresponds to a particular operation.

Even though this list (in table 3.2) is a short one when comparing to the whole list of operations currently implemented, we started with a small number for two reasons: these are, so far, the most appropriate operations related to the detection of security threats that we have access to; also, some of the operations are currently not traceable to the agent responsible, which makes them unusable in UEBA.

With the sequence of events associated with an agent in a determined period of time (*e.g.*, 24h), it should be possible to create a behavior baseline, capturing the typical rate of audit logs for each operation and each user. Any significant deviations from that baseline can indicate anomalies in the form of security threats. As an example, if a user, in a determined day, has far more login attempts or accesses to customers' personally identifiable

Table 3.3: Reference UEBA threat scenarios / misuse cases.

Name	Actor profile	Preconditions	Severity	Indicators (daily)
Account compromise	External	-	High	nr. sessions nr. countries nr. devices time of the first event
Data breach	Insider	User login User confirmed Access to contacts	High	nr. accesses to PII
Activity out of working hours	Insider	User login User confirmed	Medium	nr. calls out of working hours weekday or weekend time of the first event
Performance decrease	Insider	User login User confirmed	Low	nr. of missed calls time to answer calls session duration
Insider abuse	Insider	User login User confirmed	Low	number of calls time spent talking

information (PII) than expected, that could be indicative of a possible compromise or a data breach/theft. From the audit logs, beyond the actual audit operation, we can also collect important metadata in the security context:

- **Origin/location/network**: information extracted from the IP address allows identifying, with a certain confidence, the geolocation (latitude and longitude), country, and network associated with that address;
- **User agent**: the browser, operating system, and device information from where the event was performed.

Finally, we also extract the role of each user, typically one of the following: agent, supervisor, administrator. This is a particularly important field, as it allows to perform clustering on the contact center staff, as we expect users with the same role to have similar behavior (due to them sharing the same work), but users from different roles to have more distinct behaviors (*e.g.*, it is not expected for supervisors to have the same amount of call activity as agents do). Ideally, we should be able to create a baseline of behavior for each cluster (role), avoiding the need of profiling each user with one model per user, and detect anomalies for when users deviate significantly from the population they are inserted in.

3.2 Threat scenarios/misuse-cases

Contact center agents have access to several sensitive information while doing their daily work and, with most organizations needing to shift their staff to remote work, it may be harder to detect, without the help of a framework like the one we propose, whether agents are incurring in malicious actions, either intentionally or by accident.

The security misuse-cases defined in this section consist of some of the typical threat scenarios which we aim to detect with the UEBA framework under development. These scenarios were obtained either by contacts with actual clients, or by researching about the typical use-cases covered in UEBA products (approached in chapter 2), or adapted from the background knowledge of contact center security threats. We identified the following misuse-cases, which are complemented with the information in table 3.3, including the preconditions, the severity, and also the associated indicators of each scenario from the aggregated behavior data:

- **Account compromise**: a user account's credentials are compromised, granting an attacker illegitimate access to Talkdesk's application, or a user deliberately shares their account credentials with a third party. This scenario can be identified by comparing the hour, country, device, from which sessions of a determined user are created.
- **Data breach**: a logged-in user accesses clients' personal information at an abnormal rate, and outside of the context of the calls she is assigned. This scenario can be identified by tracking the number of accesses to contacts' PII made by each user and compare with the expected rates.
- **Activity out of working hours**: a logged-in user accesses the application during periods in which the contact center is not operating, possibly for personal purposes. This scenario can be identified by looking at user activity outside the working hours of a contact center, namely on weekends.
- **Performance decrease**: a logged-in user is not active during her period of work for long periods, exhibiting higher time to answer the calls assigned, missing several calls or closing the session earlier than expected. We can use such indicators to identify the scenario.
- **Insider abuse**: a logged-in user attempts to trick the system's reporting metrics, for instance, by answering many calls with low talk time, which could indicate the agent is not solving the clients' issues but instead aiming for personal success metrics.

Since one of the most promising capabilities of UEBA is detecting novel threats, this section serves as a guideline to important scenarios for the algorithms to detect, due to their relevance from a security and business perspective, and thus it does not intend to be an exhaustive exposition of the possible threat scenarios.

3.3 Data transformation and feature extraction

The first step of the road to obtain feature vectors to feed the anomaly detection algorithms consists of creating data transformation pipelines, responsible for ingesting the raw JSON data and transforming it in dataframes with candidate features, representing aggregated information on the behavior of users. Figure 3.1 contains the high-level steps involved in this process, which are detailed throughout this section.

The raw data we collected consists of two JSON files, one corresponding to three months of call logs, and the other corresponds to the audit logs collected during the same period, and the same agents. The "calls" file contains one JSON object per row, populated with the fields presented in table 3.1 (and some others less relevant), and each row in the "audits" file is composed of one JSON object, containing the registered operation (from the table 3.2, as well as the associated metadata.

We created two transformation pipelines, following the structure in figure 3.1, one for calls and one for audits, which transform the mentioned raw files into pandas dataframes, with

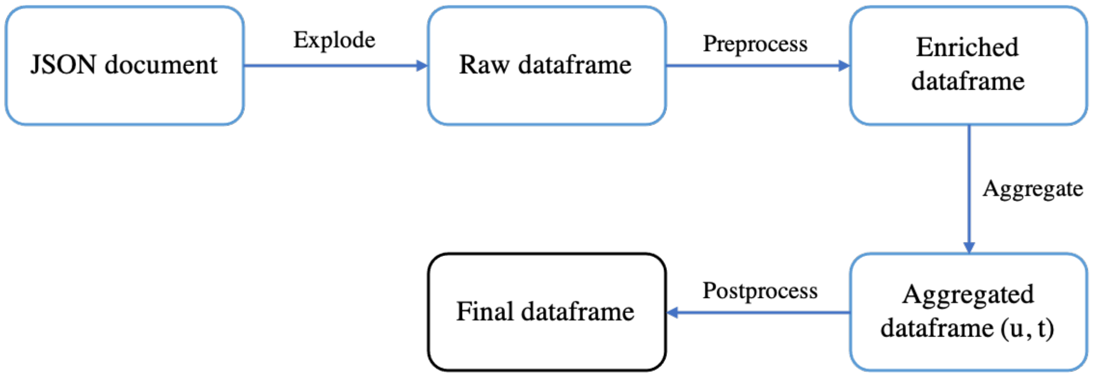
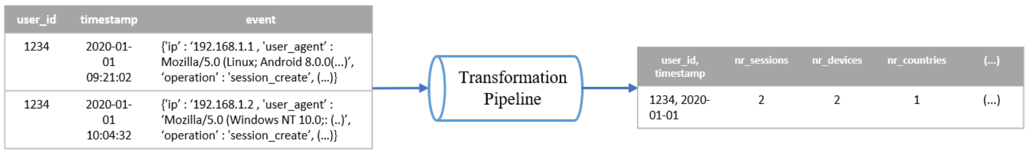


Figure 3.1: Steps performed in the data transformation pipeline



user_id	timestamp	event
1234	2020-01-01 09:21:02	{'ip' : '192.168.1.1 , 'user_agent' : Mozilla/5.0 (Linux; Android 8.0.0(...)', 'operation' : 'session_create', (...)}
1234	2020-01-01 10:04:32	{'ip' : '192.168.1.2 , 'user_agent' : 'Mozilla/5.0 (Windows NT 10.0;: (..)', 'operation' : 'session_create', (...)}

user_id, timestamp	nr_sessions	nr_devices	nr_countries	(...)
1234, 2020-01-01	2	2	1	(...)

Figure 3.2: Transformation pipeline example for the audit logs (with synthetic events)

each row corresponding to the activity of one user during a particular period (*e.g.*, 1h, 3h, 8h, 24h, etc.). For the audit logs, the following steps are performed during the pipeline:

1. Explode (or parse) event: the JSON file is transformed into a dataframe by using the schema of the data;

2. Preprocess: prepares the dataframe obtained in the previous step to the aggregation step, by removing rows with errors or null values (does not happen frequently in this dataset), perform type conversions on certain fields, derive new fields to utilize more group functions after the aggregation, convert the timestamp of each event into the pandas' timestamp format;

3. Aggregate: we group the events by the *user_id*, which is followed by re-shaping the dataframe according to the time period parameter t (the default value is 24h, which we use in this work). This creates a multi-index aggregated object, to which we need to apply the appropriate group or aggregation functions to summarize the activity of a user in a particular period. The possible functions include sum, count (or count distinct), mean, median, first, last, max, min, and in some cases, we apply multiple functions to the same field (as present in tables 3.4 and 3.5). Each row in the resulting dataframe contains a multi-level index composed by an *user_id* and a *timestamp*, with values corresponding to the multiple aggregation strategies mentioned.

4. Postprocess: The final step consists of preparing the dataset for exploratory analysis. We rename some of the columns, to properly express the aggregation functions used, drop the empty rows inserted when re-sampling, and other minor changes.

Figure 3.2 contains an example of the transformations specified above. After being sent through the pipeline, the JSON objects are transformed into the dataframe with summary statistics of user activity. This process is analogous for the call data, except evident differences such as the JSON schema and fields extracted.

Regarding the time period defined for the aggregations, **we selected 24h** as the most appropriate bucket. The decision was taken due to the impossibility of properly defining a

Table 3.4: Features extracted from the aggregated audit logs

Candidate Feature	Description	Extracted from	Aggr. function
nr_ip_addresses	No. different IP addresses registered	IP address	distinct count
nr_networks	No. different networks the user logged-in from	IP address	distinct count
nr_usr_agents	No. different user agents the user used	user agent	distinct count
nr_countries	No. distinct countries the user logged-in from	IP address	distinct count
nr_sessions_created	No. sessions created (number of login events)	operation	count
nr_contact_reads	No. access to contacts/PII performed	operation	count
nr_pw_changes	No. password changes performed	operation	count
nr_recordings_accessed	No. call recordings accessed	operation	count
first_event_time	Time of the 1st event of the day	timestamp	first
last_event_time	Time of the last event of the day	timestamp	last

Table 3.5: Features extracted from the aggregated call logs

Candidate feature	Description	Extracted from	Aggr. function
nr_calls	Number of daily calls (inbound or outbound)	- (raw)	count
inbound_call_rate	Percentage of received calls	Call direction	sum
total_talk_time	Total time spent talking	Call talk time	sum
median_talk_time	Median time (per call) spent talking	Call talk time	median
avg_talk_time	Average time (per call) spent talking	Call talk time	mean
nr_missed_calls	Number of calls missed by the agent	Call missed	sum
avg_waiting_time	Average time customers waited for calls to be answered	Call waiting time	mean
median_waiting_time	Median time customers waited for calls to be answered	Call waiting time	median
first_call_time	Time of the first call of the day	Call start time	first
last_call_time	Time of the last call of the day	Call start time	last
avg_call_cost	Average cost spent in calls	Call cost	mean
median_call_cost	Median cost spent in calls	Call cost	median
max_call_cost	Maximum cost spent in a call	Call cost	max
total_calls_cost	Total cost spent in calls	Call cost	sum
calls_out_working_hours	Number of calls performed out of the contact center's working hours	Call inside working hours	sum

session as the unity (which would be, in our perspective, another appropriate approach), as the time for the beginning and end of a session is not well defined in our data. Furthermore, the decision also has in account other UEBA products that use the same bucket (such as [20]), and the fact that sessions do not cross multiple days (the contact center data used in this work is not opened 24h a day).

Tables 3.4 and 3.5 contain the features extracted, resulting from the transformations performed, for audit logs and calls, respectively, as well as a description of each feature, the original raw field where it was extracted from and the aggregation functions used. We also introduced the field "approx_session_duration", calculated by subtracting the timestamp of the first event of a day to the last one, giving an approximate duration of a determined session.

The final step of the transformation step consists of joining the final calls and audits dataframes. This is necessary as we aim to create one feature vector representative of user behavior that is as complete and diverse as possible. Moreover, we want to be able to correlate call-related candidate features (such as the number of calls in a day) with audit

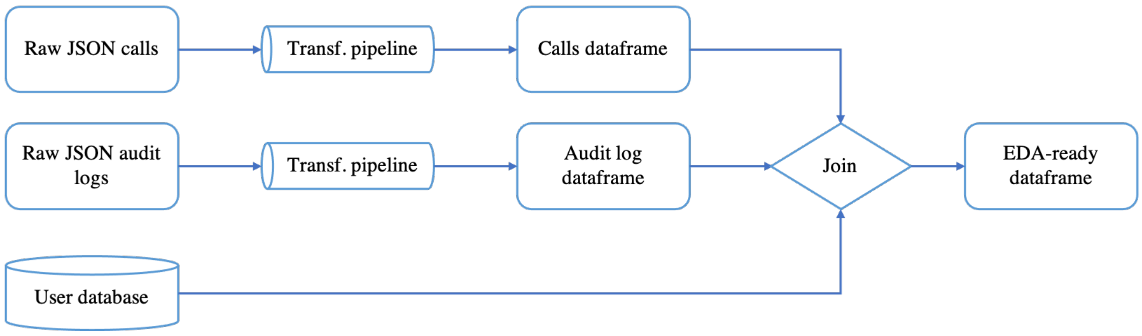


Figure 3.3: Full process to obtain one single dataframe, ready for EDA, from the multiple sources.

log candidate features, such as the number of accesses to PII. Thus, we want to be able to summarize the entire activity of a user concerning calls and audits in a single row, reason why an outer join is necessary on the dataframes.

Figure 3.3 contains the whole transformation process, from the JSON files to the final dataframe, which will be used for exploratory data analysis (EDA) and feature selection. The figure introduces the join operation performed over the indexes of the dataframes resulting from the calls and audits, as well as the "user database", from where we extract the user roles.

3.4 Exploratory data analysis and feature selection

Selecting the best/most appropriate set of features for anomaly detection is one of the biggest challenges in this project. In supervised learning, having a target makes it easier to perform feature selection. However, the unsupervised nature of our problem hinders the decision to discard or include features or even to assess the necessity of creating new features [19]. Hence, we will use several approaches to select from the candidate features:

- **Background knowledge**: we use knowledge from the security domain on which fields are relevant for detecting the threat scenarios defined in section 3.2, based on the indicators for each scenario;
- **Cluster analysis** concerning the expected behavior/activity differences between each of the roles in the contact center;
- **Univariate analysis** and the kurtosis measure: we analyse each feature individually, by using plots (such as the feature's distribution with histograms, box-plots, violin-plots) and summary statistics (such as the mean, standard deviation, maximum and minimum values, quartiles, and others). The kurtosis measure, as suggested in [19], identifies distributions containing heavier tails;
- **Correlation analysis**: we analyse the Pearson correlation between each pair of features, which gives fundamental insights on events that are performed with some relation to others. It may allow us to create new features that explicitly represent the correlation between two features (*e.g.*, if feature A and B are strongly positively correlated, we could use a ratio feature R, where R = A / B).

Before beginning with EDA, we removed the null values introduced by the join operation performed in the previous section (*e.g.*, days where users performed audit operations but

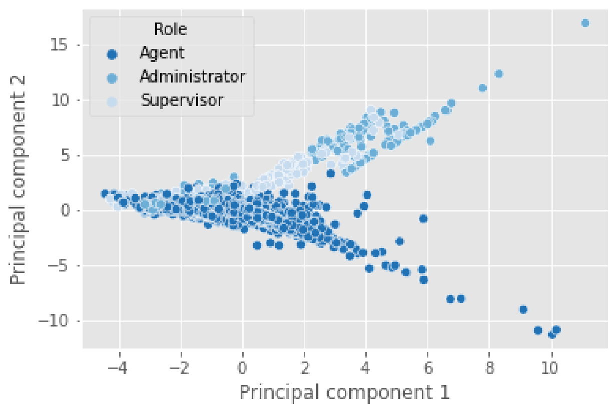


Figure 3.4: Scatter-plot of the dataset based on user roles, using PCA with 2 components (with a summed explained variance of 64%)

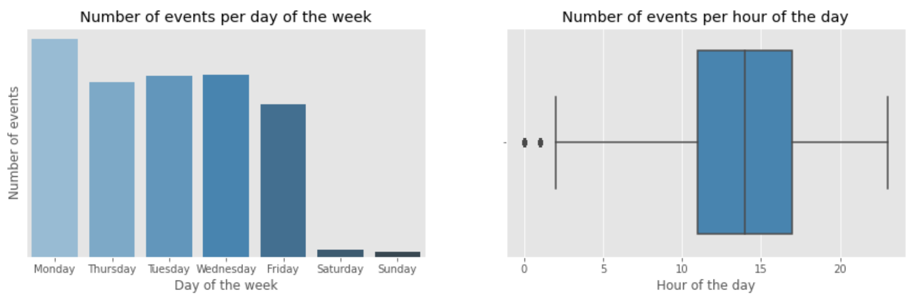


Figure 3.5: Number of events for each day of the week (plot on the left) and for each hour of a day (plot on the right)

no call activity leads to all fields related to calls being null in that day, or vice-versa). As all these null values correspond to the absence of activity by the user, the logic imputation strategy is to replace the null values with 0.

We started this analysis by exploring the aggregated activity for each role a user can have. Figure 3.4 contains the activity of users, obtained with the use of a subset of features and principal components analysis (PCA), retaining the 2 principal components, after scaling the aggregated audit and call data. The existence of three clusters is evident, one for each of the roles present.

Due to the evident differences of behavior between the different roles in the contact center, depicted in figure 3.4, this step of feature selection should be done individually for each role, and different conclusions would be drawn from each analysis. Therefore, we decided to perform this first analysis (used in this work) for the agents, and replicate it as future work for the other roles, as the agents are the group that contains the most individuals and that represents the higher activity in the contact center.

Univariate analysis

Due to the importance of time for some of the threats defined, we started the univariate analysis by investigating how the events are distributed during each day of the week, as well as during each hour of the day. This information is shown on figure 3.5. From the plot on the left, we identify that the activity of the contact center is distributed almost uniformly during the weekdays (except for a little more activity on Monday and a little less on Friday), but there is almost no activity on the weekends. This suggests that activity on Saturdays or Sundays could be suspicious, and can be registered through the usage of a boolean feature indicating if the day of the event is a Saturday or Sunday. From the plot on the right, represented with a box-plot due to it having less information than the distribution histogram (privacy concerns), we identify that 50% of the activity is registered between the period after 10:00 and somewhere around 17:00, as well as some outliers around midnight, confirming how important the time is for detecting activity outside the regular working hours.

To proceed with the analysis of the individual features, we used both qualitative and quantitative analysis of the distributions, including histograms, box-plots, and statistical measures such as the kurtosis.

The kurtosis measure (K) identifies the presence of heavy tails in a statistical distribution. High values of K indicate a very non-uniform distribution, which identifies features that may be more prone to containing extreme values that could translate into suspicious behaviors. To calculate K for each feature, we start to calculate the mean μ and standard deviation σ of that feature's values, and standardize it, as shown in equation 3.1. After that, the kurtosis K is calculated using equation 3.2, where N equals the number of data points.

$$z_i = \frac{x_i - \mu}{\sigma} \tag{3.1}$$

$$K(z) = \frac{\sum_{i=1}^{N} z_i^4}{N} \tag{3.2}$$

Even though we are not able to show the visual representations, due to privacy concerns associated with disclosing the statistical distributions of the fields, we present the main conclusions of this step below (we refer to several features from tables 3.4 and 3.5):

- None of the data distributions resemble the normal distribution, which is an issue for algorithms that assume a normal distribution of the data, such as the Mahalanobis method. The distributions are typically right-skewed with several 0 values, corresponding to the absence of activity;
- The distributions and quartiles show significant similarities between the overall distribution of the number of calls ("nr_calls") and the number of contacts read ("nr_contact_reads"), with the latter having more extended outliers (heavier right tail), and thus a higher kurtosis value. This relation should be further explored in the next step (correlation analysis);
- Features related to the event times (*e.g.*, timestamp of the first event of the day) contain outliers exposed by the quartile values around midnight, suggesting activity out of the working hours of the contact center;
- The number of missed calls ("nr_missed_calls") and number of calls out of working hours ("nr_calls_out_working_hours") are the fields that exhibit the highest kurtosis values (around 386 and 3047, respectively). Both the fields have a mean value close to 0 but significant outliers (*e.g.*, more than 100 missed calls in one single day), probably indicating these are two strong candidate features to select and use in the models;

- The fields related to the creation of sessions ("nr_sessions_created", "nr_countries", "nr_networks") reveal a typical usage of one single concurrent session, with exceptions of up to 11 different sessions created. The number of countries and number of networks seem to have a similar distribution with most of the values being equal to 1, as expected in a typical situation. These fields should probably not be included in this multivariate approach, as a separate and univariate analysis of each of them, or even a rule-base approach, would probably suffice;
- The feature "total_talk_time" contains the highest variability, with a near-uniform distribution for values between 0 and 20.000, and consequently the smallest kurtosis value. These factors indicate that such feature may not be particularly interesting to be used by itself for anomaly detection;
- All the values related to access to call recordings are equal to 0, due to the agent role not having permission to access such sensitive information, reason why such fields should be excluded from further analysis.

The univariate analysis step allowed us to obtain insights on all the fields in the dataset. We were able to identify fields to include in the evaluation as well as fields that should be removed. The next step consists in analysing pairs of features to identify potential correlation patterns.

Correlation analysis

Often, in simple supervised learning approaches, we look for features that by themselves or together with other features can achieve high correlation values with the labels. In an unsupervised scenario like this one, we can analyze the correlation between pairs of candidate features, hoping to find interesting feature dynamics and to derive new features that can ease the identification of anomalies.

For correlation analysis, we start by utilizing a heatmap of Pearson correlation for several pairs of features from the final dataframe containing the joined sources, presented in figure 3.6, and with it we are able to extract valuable insights:

- The most interesting correlation (from a security perspective) happens between the number of calls ("nr_calls") and number of contact reads, representing access to clients' PII ("nr_contact_reads"). It is possible to assume that the access to PII is highly related to the number of calls an agent performs, meaning that an excess number of contact reads when compared to the number of calls (the ratio) could indicate an anomaly. This ratio could be used as a feature, enforcing the importance of this correlation.
- The positive correlation between the number of calls ("nr_calls") and the approximate session duration ("approx_session_duration") shows that the longer the session, the higher the number of calls an agent performs, which could imply that agents with long session times and few calls could be suspicious.
- There is a really strong positive correlation between the number of calls and the respective cost ("total_calls_cost") from those calls, as expected.
- Since the fields derived from the sessions created ("nr_countries", "nr_devices", "nr_networks") are close to being colinear, we will be inclined to remove them on account of speed and simplicity.

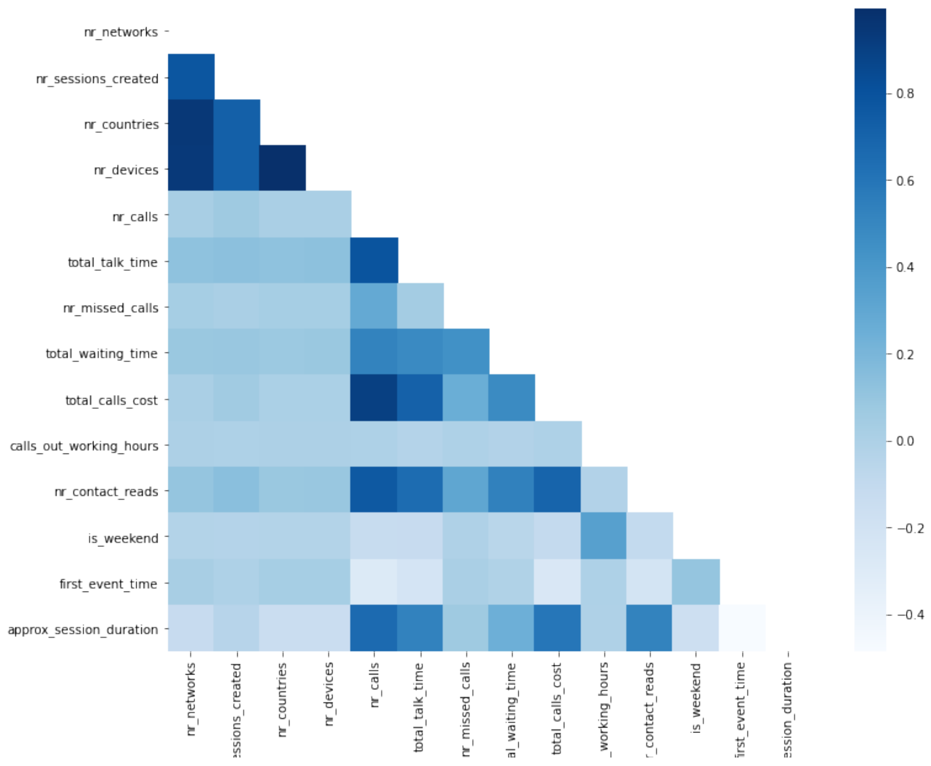


Figure 3.6: Pearson correlation between pairs of example features extracted

The last step of correlation analysis consisted of using seaborn's pair plots, which display scatter plots for pairs of features and the features' distribution in the diagonals. Even though it was not possible to include the plots in the document (due to privacy concerns), the results enforce the correlations present in the heatmap from figure 3.6, through the visible linear relation between pairs of fields such as "nr_calls" and "total_calls_cost" (positive correlation), "nr_calls" and "nr_contact_reads" (positive correlation), "first_event_time" and "approx_session_duration" (negative correlation).

The analysis conducted in this section allowed us to obtain several important insights and assumptions regarding the feature-set available for the definition of a behavioral profile. These conclusions/assumptions **do not exclude any features from being selected in the algorithms**, but are rather just an assumption of some of the most promising fields in the data.

This page is intentionally left blank.

Chapter 4

Algorithms

This chapter contains a description of the algorithms selected for anomaly detection and for the interpretability task. Also, we present the final dataset (which resulted from the work presented in the previous chapter) used in most of the steps of the validation process.

It is important to mention that all the algorithms used in this work, both for detecting and interpreting the anomalies, are available in recognized python libraries, such as scikit-learn and pyOD, or through official Github repositories. As such, the time and effort required for us to be able to use the algorithms was not as extensive as the time spent on the analysis of the results, which is presented in the next chapter. Nevertheless, details of the algorithms, such as the parameters or important assumptions made are presented in this chapter.

4.1 Chosen anomaly detection algorithms

This section contains the details of the anomaly detection algorithms chosen for our problem. The selection was made based on the results of the analysis present in section 2.2.1, and we decided to include a total of 5 algorithms, with different characteristics, presented below. Table 4.1 contains a description of each algorithm, with information such as the python library from where we imported the algorithm, the parameters it requires, and the way the anomaly score is obtained. The following subsections contain a more detailed view on each algorithm.

4.1.1 HBOS

Presented in section 2.2, histogram-based outlier score (HBOS) is a statistical algorithm, which assumes feature independence. We expect it to be the most scalable with regards to fitting and prediction time.

An implementation of HBOS is available in the library pyOD, which is used in this project. The threshold between a normal instance and an anomaly is calculated based on the parameter "contamination", present in all the algorithms under evaluation. The algorithm is also parameterized by the number of bins in the histogram built. In this work, we use the default value (10) and do not vary the parameter due to the lack of possibility of tuning the parameter for each of the clients we intend to deploy the framework on.

The anomaly score of this implementation is calculated based on the density of the bin that each feature value falls into, and for reasons of consistency with other algorithms in

Table 4.1: Overview of the anomaly detection algorithms

Algorithm	**Library**	**Parameters**	**Anomaly score**
HBOS	pyOD	contamination n_bins	probability of the event
Mahalanobis method	scikit-learn	contamination	Mahalanobis distance
Isolation forest	scikit-learn	contamination n_estimators n_samples	average depth
PCA	pyOD	contamination n_components n_selected_components	distance to hyperplane
Autoencoder	pyOD	contamination hidden_neurons hidden_activation output_activation loss optimizer epochs batch size dropout rate l2 regularizer	reconstruction error

pyOD, the bigger the score, the higher the abnormality of the instance.

4.1.2 Mahalanobis method

The second algorithm we chose is the Mahalanobis method, a statistical algorithm presented in section 2.2.1 with two main advantages: it is a simple algorithm to implement and at the same time powerful when compared to other statistical methods, because it computes distances using the covariance matrix, which allows having a simpler but still multivariate anomaly detection algorithm. Plus, this method is used successfully in a real user and entity behavior analytics (UEBA) implementation [20]. The biggest disadvantage consists of the assumption that the data has a multivariate Gaussian distribution, which is not the case in our dataset, as we concluded in chapter 3.

An implementation of the Mahalanobis method is available in scikit-learn, which uses one parameter, "contamination", widely seen in anomaly detection algorithms from this library. It starts by calculating the Mahalanobis distance for all the elements in the data, and uses the contamination to establish a distance offset that separates inliers from outliers. At test time, the distances are calculated for the test examples, and each of them is classified as an inlier or outlier based on whether the distance is smaller or bigger than the offset, respectively.

4.1.3 Isolation forest

Isolation forest is one of the most popular algorithms when it comes to unsupervised anomaly detection. The method, explained in detail in section 2.2.1 is fast, powerful and an implementation is available in the popular python library scikit-learn. Similarly to the implementation of the Mahalanobis method, the isolation forest in scikit is parameterized by the contamination rate, but also by the number of estimators (number of trees in the ensemble) and the maximum number of samples to draw from the training dataset for each estimator, which defaults to 100.

At training time, the algorithm creates the ensemble of estimators which isolate the anomalies as leafs close to the root of the tree. The anomaly score is calculated for each instance in the dataset, using the mean anomaly score of the trees in the ensemble. The anomaly score for each tree is given by the depth of the instance in that tree (the lower the score, the more abnormal de instance is). Afterwards, the contamination rate is used to create an offset anomaly score, which is subtracted in test time to the anomaly score of the test instance, resulting in a classification of inlier or outlier whether the difference is bigger or smaller than 0.

4.1.4 PCA

Principal components analysis (PCA) is another algorithm successfully applied in an UEBA implementation [43]. In the anomaly detection library pyOD, an implementation of PCA is available, which is based on [44] and [19].

The implementation starts by standardizing the data, proceeding to the projection of the training matrix to a subspace that retains a determined amount of variance, typically 95% or 99% (we went with 95%, after testing both options). Afterwards, the Euclidean distance between a determined instance and the projection hyperplane (given by the eigenvectors) is calculated, weighted by the eigenvalues of the selected components, as presented in equation 4.1, where *dist* corresponds to the Euclidean distance, μ corresponds to the sample mean, e_j to the jth eigenvector with eigenvalue e_j and d to the number of principal components (eigenvectors).

$$Score(X) = \sum_{j=1}^{d} \frac{|dist(X - \mu, e_j)|^2}{\lambda_j} \tag{4.1}$$

Higher distances result in more likelihood of the instance being classified as an outlier, threshold that is given by the contamination parameter specified.

This implementation of PCA **tends to be robust** to some training contamination (outliers in the training set), as the score associated with an instance is calculated with regard to an optimal hyperplane, instead of a particular value [19].

4.1.5 Autoencoder

The autoencoder is a neural network with a bottleneck architecture, as explained in section 2.2.1. An implementation of the algorithm is available in pyOD, which we use in this work. After standardizing and shuffling the data, the model is built using the Keras API, by creating a sequence of "Dense" (fully connected) layers with the dimensions specified as a parameter. The following parameters were chosen for the network:

- The dimensions of the network are specified as a list of the number of neurons for each layer of the encoder (we set the parameter to [8, 4, 2], resulting in a neural network with 6 hidden layers, with the number of neurons in equal to 8, 4, 2, 2, 4, 8);
- We use the default values for the **activation function** of the hidden layers (ReLU) and also for the output layer (sigmoid);
- We also use the default optimizer approach (adam) and loss metric (mean squared error);
- Regarding strategies to control overfitting, we use the default L2 regularization value (0.1), as well as the dropout rate (0.2);
- We train with a batch size of 32 instances and for 100 epochs.

The contamination parameter is used to calculate a threshold for the reconstruction error, which is obtained by subtracting the original instance to the reconstructed one. In test time, if the reconstruction error of an instance is higher than this threshold, it is classified as an anomaly.

As an important consideration, we should mention that we will not, at least in the first iterations of the framework in production, have an autoencoder version in production, even if it yields the best results. However, we thought it would be interesting to compare the algorithms above with a neural network algorithm, and assess if we can reach the stability of the model, if it is hard to train and if the results are significantly better than the other algorithms.

4.2 Chosen interpretability algorithms

As stated throughout the document, it is of utter importance for us to be able to provide context along with each anomaly or, more specifically, which were the features that contributed the most to a determined alarm. In this section, we go through the implementation details of the three approaches which we will compare to achieve this important goal: z-score relative importance, depth-base feature importance for the isolation forest (DIFFI) and Shapley additive explanation (SHAP).

Z-score relative importance

In [20], the authors address the **problem of interpretability** with a simple strategy: they start by calculating the z-score of a determined test instance, using equation 3.1, where μ is the sample mean vector and σ is the sample standard deviation vector along each dimension. Afterwards, the importance of each variable j in the vector z is given by equation 4.2, where M is the number of dimensions in the dataset.

$$imp_j = \frac{z_j^2}{\sum_{i=1}^{M} z_i^2} \tag{4.2}$$

This solution is straightforward to implement and apply in our data, and has the advantages of being totally independent of the algorithm used for anomaly detection, as well as being scalable due to its simplicity. On the other hand, the algorithm may struggle when an anomaly is the consequence of an abnormality in the correlation between two features, such as the case of the number of calls vs. the number of accesses to PII ("nr_contact_reads"), with a strong positive correlation (figure 3.6), as the importance of each feature is calculated independently.

DIFFI for the Isolation Forest

DIFFI [32], was mentioned in section 2.2.2, and the authors made available an implementation in python, with a reference present in the publication.

In test time, we use the algorithm to calculate the importance of each feature in each instance in a determined batch. The function *local_ diffi_ batch* present in the algorithm's utilities file receives as input the fitted isolation forest object and the batch of examples to interpret, and returns three different structures:

- a matrix containing in each row the interpretability score of each feature in each row;
- a matrix containing in each row the ranked features ordered in decreasing fashion for each instance;
- the execution time of the batch.

DIFFI is a model-specific interpretability algorithm, meaning it can only be used to explain instances classified by the isolation forest.

Table 4.2: The different feature sets used

FS	Description	M (#)	Features
1	Conservative feature set obtained from background knowledge and EDA	8	nr_sessions_created, nr_calls, nr_missed_calls, avg_waiting_time, calls_out_working_hours, nr_contact_reads, is_weekend, first_event_time
2	Extended feature set with features excluded from FS1	13	FS1 + total_calls_cost, approx_session_duration, nr_devices
3	Extended feature set with additional features engineered	15	FS2 + call_cr_ratio, call_session_ratio

SHAP values

SHAP can be used to explain the output of any machine learning model, and has a python implementation available in [45].

The repository contains SHAP applications to several types of models, such as tree ensembles and deep learning. In our case, we use the **KernelExplainer**, which uses linear regression to estimate the SHAP values, and is model-agnostic, and **TreeExplainer**, which we use for interpretability of the isolation forest, which leverages the tree-based structure of the anomaly detection algorithm.

We applied the model-agnostic variant of SHAP on different models (Mahalanbois method, PCA and autoencoder), and the TreeExplainer on the isolation forest. The results of the analysis are presented in the next chapter.

4.3 Training data

As mentioned in the previous chapter, our dataset consists of three months of real data, collected from user activity in the Talkdesk application. After the transformation pipelines presented in the previous chapter are ran on the original data, the final dataset contains, in each row, the activity of one user U in a period of 24h inside the period of three months. As such, we divided the dataset so that the number of training rows N is given by the instances belonging to the first 60 out of the about 85 days for training, leaving the latest 24 days for testing/validation purposes (about 30%).

Regarding the number of dimensions, M, we will evaluate the framework on three distinct feature sets, referred to as FS1, FS2, and FS3, specified in table 4.2. These feature sets were chosen with the help of background knowledge along with the exploratory data analysis (EDA) step performed in the previous chapter, with FS1 being the most conservative (only what we believe to be the most important features), and the subsequent sets being extensions of the previous, by adding some other interesting features to FS1, in the case of FS2, and adding engineered features, in the case of FS3. During the validation process, both the number of training rows and features vary according to different settings we evaluated, such as the number of days used for the baselines (14, 30 and 60 days are the possible settings), and the feature set used (we experiment with the 3 presented).

Chapter 5

Evaluation and deployment

Validation in unsupervised problems is another known challenge, as it is not directly possible to compute metrics such as accuracy, precision, recall, or others, as we do not have readily available labeled datasets. Nevertheless, validation is probably the most important and demanding step of this work, as a deployment in a production environment for several customers implies confidence in the correctness of the outputs obtained.

We divided this chapter into two distinct parts, evaluation and deployment.

The testing steps conducted for the implementation of the extract, transform, load (ETL) and the transformation pipelines are out of the scope of this work.

5.1 Evaluation

In this section, we present the steps conducted to evaluate the results of the framework proposed, using the algorithms and the dataset presented in chapter 4.

The results of this analysis will determine the main settings to use in the staging deployment, presented in the next section.

This section is divided into 4 different steps: **generation of anomalous instances**, used in some of the subsequent steps; **validation of the threat scenarios**, by using the labeled dataset to evaluate the results of each individual model using supervised classification metrics; **sampling from supervisor activity**, an approached based on inspiration from other user and entity behavior analytics (UEBA) solution [43], consisting in sampling instances from roles that we expect to exhibit different behavior than agents and use supervised classification metrics to evaluate the algorithms; **evaluation of the interpretability methods**, by leveraging the labeled dataset created to assess whether the algorithms are capable of identifying the most relevant feature(s) in a determined anomalous instance.

5.1.1 Generation/labeling of anomalous instances

The first strategy implemented to help assessing the results of the algorithms adopted consisted in synthesizing malicious activity, by creating and labeling instances of aggregated activity corresponding to the threat scenarios presented in table 3.3. Even though this is a manual laborious task, it will allow us to use the metrics typically used in supervised classification problems.

Due to the laborious nature of creating labeled sequences of events that correspond to anomalous sessions, we decided to go with a collaborative approach, where several members with both security and product-related knowledge in the team contributed with their own examples of anomalies.

In order to enable other team members to contribute, we provided them with a spreadsheet containing a template with all the metadata necessary to create such sessions. To facilitate the work as much as possible, the spreadsheet contained:

- the columns ready to fill, including a column for each of the fields present in FS2, but also a column for the fields that explain the anomaly and the threat scenarios associated;
- example anomalies already filled, to facilitate the understanding of the activity;
- a **template of a typical audit log and call log**, cleaned to only contain the fields necessary (*e.g.*, fields such as IDs or others less relevant to the anomaly detection task were removed);
- the **summary statistics** of each field in our dataset (mean, standard deviation, quartiles, minimum and maximum values), as well as box plots;
- a correlation heatmap and pair plots of fields with **significant correlations** to highlight how the fields relate;
- a **description of each threat scenario** defined, present in table 3.3, along with an example anomalous session (sequence of events) for each of the 5 scenarios.

After going through the spreadsheet with the team members, their task included creating anomalous instances of aggregated activity, specifying the values for each of the aggregated fields present in FS2 (table 4.2), and also relevant metadata of each instance, including **the feature(s) that cause the anomaly**, used to evaluate the interpretability methods, as well as the **threat scenario(s)** associated, if any, allowing us to detect if the algorithms are equally successful in detecting different types of threats.

A total of 4 members contributed to the creation of heterogeneous anomalies, resulting in about 150 instances, with different degrees of abnormality. Figure 5.1 shows the distribution of anomalies for each of the threat scenarios defined in table 3.3, showing the variety achieved. One example of an instance related with the threat scenario "Account compromise" would be an agent with, for instance, multiple sessions created in the same day from multiple countries (feature "nr_countries"); an example of a "data breach" would be a number of accesses to personally identifiable information (PII) that is both higher than the user normally does, and also significantly higher than the number of calls performed by the same agent; and so on.

5.1.2 Validation with the labeled examples

As mentioned in section 4.3, the dataset was split into train and test sets, with the instances corresponding to the first 60 days being used for training. The testing set consists of the latest 24 days in the data, which correspond to about 30% of the 85 days in the collected dataset.

The task conducted in this subsection consisted in fitting each of the algorithms selected with the training set (using all of it or just a part, depending on the baselining period

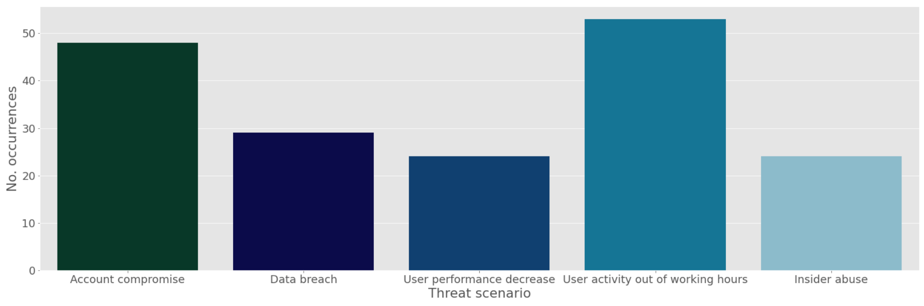


Figure 5.1: Use-case distribution in the synthetic instances

setting, explained ahead in the section), and use the test set, after ensuring no outliers are present, joined with the labeled instances, to evaluate each algorithm using supervised classification metrics (F1-score, AUC-ROC, precision, recall).

The test set was manually analysed regarding the presence of outliers coming from the contamination present in the data, by using box and scatter plots to identify outliers, and all of the resulting 688 'normal' instances were labeled with the negative class. On the other hand, the malicious instances resulting from the manual labeling process described in the section above (150 instances) were assigned the positive class.

We performed experiments with several algorithms, along with different varying settings, namely:

- The **baselining period**, corresponding to how many days of historical data are used to fit the algorithm. The possible values are 14, 30, or 60 days (2 weeks, 1 month, and 2 months), from which we expect to understand how much the period of historical data used to train the model affects its capability of detecting significant anomalies. We assume 14 days to be the minimum from which it is possible to build a significant baseline, but we expect more accurate results with more historical data;
- The **feature set** applied, between FS1, FS2 and FS3, as presented in table 4.2. We assume that selecting the right set of features is of utter importance in the results, but due to the unsupervised nature of the problem, we need to experiment with different sets;
- **Training contamination**, which we attempt to vary by removing some of the anomalous instances in the training data, using the robust PCA introduced in section 4.1.4 to score each instance, and remove instances with a score higher than a determined percentile. We defined three settings for contamination reduction, namely "true", which corresponds to preserving the original contamination (no reduction is performed); and the settings "med" and "max", which correspond to removing the instances with the highest 1% and 5% anomaly scores;
- The **contamination parameter**, the parameter used to estimate the percentage of outliers present in the training data, present in all the models applied. We experiment with the values 2, 5, and 10 percent of estimated contamination.

We fit an instance of each of the 5 algorithms selected for each combination of the settings described, leading to a total of **81 versions of each model** (3 baselining periods, 3 feature sets, 3 contamination reduction settings and 3 values for the contamination parameter),

Table 5.1: The 4 settings of each algorithm with the highest F1-score, using the labeled test set, sorted by decreasing order of the metric.

Algorithm	Training set parameters			Model parameters	Metrics					
	Baseline period	Feature set	Training contam.	Contam.	F1	ROC	Prec.	Rec.	FP	FN
Autoencoder	30	FS2	true	0.05	0.947	0.973	0.935	0.96	10	6
Autoencoder	14	FS1	true	0.05	0.944	0.969	0.934	0.953	10	7
Autoencoder	60	FS2	true	0.05	0.944	0.972	0.929	0.96	11	6
Autoencoder	14	FS1	med	0.05	0.936	0.963	0.933	0.94	10	9
PCA	14	FS1	med	0.05	0.932	0.951	0.951	0.913	7	13
PCA	60	FS2	true	0.05	0.931	0.964	0.916	0.946	13	8
PCA	14	FS2	max	0.05	0.93	0.958	0.927	0.933	11	10
PCA	14	FS1	true	0.05	0.93	0.958	0.927	0.933	11	10
Mahalanobis	14	FS1	true	0.05	0.893	0.915	0.954	0.839	6	24
Mahalanobis	60	FS1	true	0.05	0.887	0.914	0.94	0.839	8	24
Mahalanobis	60	FS2	true	0.02	0.876	0.899	0.96	0.805	5	29
Mahalanobis	14	FS2	true	0.02	0.872	0.896	0.96	0.799	5	30
IForest	14	FS2	true	0.1	0.861	0.927	0.831	0.893	27	16
IForest	14	FS1	true	0.1	0.851	0.915	0.838	0.866	25	20
IForest	14	FS2	true	0.05	0.849	0.881	0.943	0.772	7	34
IForest	30	FS1	true	0.05	0.847	0.89	0.902	0.799	13	30
HBOS	30	FS1	max	0.02	0.757	0.831	0.837	0.691	20	46
HBOS	30	FS2	med	0.02	0.756	0.809	0.959	0.624	4	56
HBOS	14	FS1	true	0.02	0.741	0.806	0.912	0.624	9	56
HBOS	14	FS1	max	0.05	0.74	0.829	0.788	0.698	28	45

and thus a total of 405 instances of models trained. In Appendix C we present the results obtained for all the trained instances, with the respective settings and classification scores.

Table 5.1 contains the 4 results with the highest F1-score between each of the 81 versions of each algorithm, presented in decreasing order of the metric. With this information, we are able to draw some conclusions regarding the impact of the algorithm, the baselining period settings, the feature set applied, and the contamination present in the training set, which are all factors of extreme relevance based on the need of generalization of this approach to other clients.

Impact of the algorithm chosen

As expected, the machine learning algorithm applied has a strong impact on the results obtained. We used, in this study, 5 different algorithms with different structures and assumptions about the data. In fact, in table 5.1, where the 4 results with the highest F1-score of each of the algorithms is presented, we notice that the best 4 versions of the autoencoder have a higher F1-score than the highest score obtained with PCA, the 4 highest scored versions of PCA result in a higher score than the best setting of IForest, and so on. **This shows that the algorithm chosen is critical in obtaining more accurate results**, even when we vary several training set parameters, as the table shows.

Figure 5.2 present the AUC-ROC curves of the top-performing (highest ROC score) instance of each of the models.

As, to some extent, expected, the autoencoder is the algorithm that achieves the more consistent results and highest scores. However, surprisingly, the PCA implementation appears very close to the neural network approach, with about 0.95 in its highest ROC-score setting. This version of PCA is characterized by some robustness and scalability,

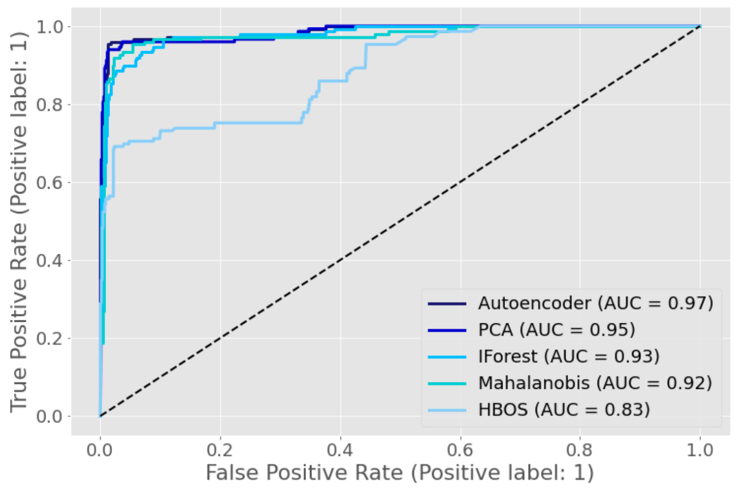


Figure 5.2: AUC-ROC curves for the version of each model with the highest ROC score.

as mentioned in the previous chapter, making it one of the strongest candidates for the production deployment.

The highest ROC score obtained with the isolation forest (IForest) is 0.927, explained by several false positives, when compared with the highest scored principal components analysis (PCA).

The best instance of the Mahalanobis method obtained a ROC score of 0.915, and a significant increase in the number of false negatives, when compared to the PCA, which, due to the similarities between the algorithms discussed in section 2.2, is most properly explained by the strong assumption the former makes on the distribution of the data (assumes a multivariate gaussian distribution).

Finally, the histogram-based outlier score (HBOS) achieves a maximum ROC score of 0.83, which shows that, in our specific problem, the relations between the features are relevant in determining whether instances are outliers or not, and thus they should not be modeled independently.

Baseline period impact

The baseline period defines how many days of data are used to build the behavior baselines. This parameter is extremely relevant to answer two questions: when processing data from a new client, how many days do we need to wait to be able to build significant baselines; and, does a two-month baseline improve significantly the results when comparing to a model trained with a baseline of 1 month, requiring to train with a double amount of data.

The response to both questions was obtained by varying the baseline duration parameter between 14 (the minimum number of days we defined), 30, and 60 days. This results in, respectively, 648, 1508, and 3044 training instances in our dataset.

Figure 5.3 contains, on the left, box plots of the F1-score distributions for each model, when varying the baseline duration setting and, on the right, the number of occurrences of each setting in the top 20 performing models, independently of the model. Also, table 5.1 presents the baseline period (second column) used in the top-performing (highest F1-score)

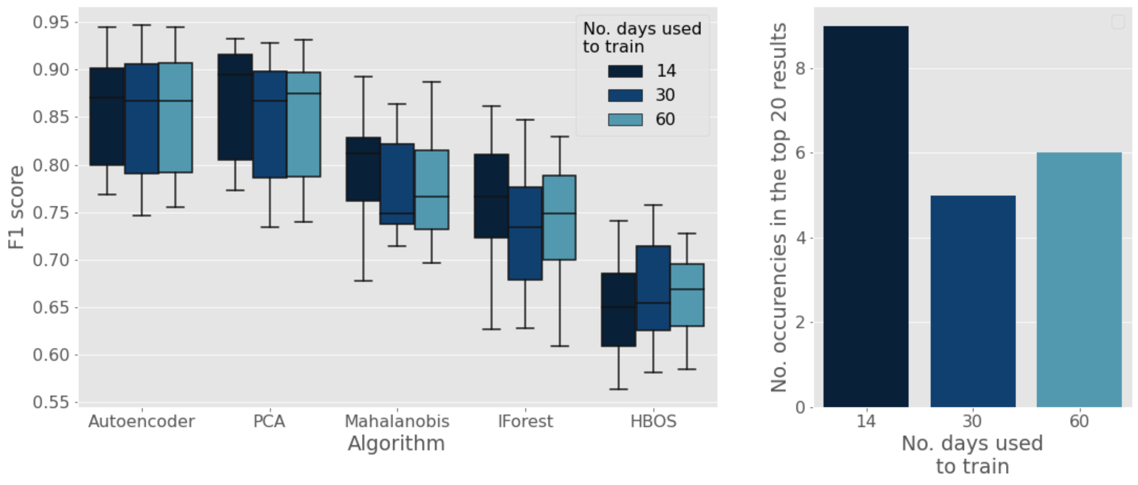


Figure 5.3: F1-score distribution for each algorithm and each value of the number of days used to build the behavior baselines, for all the different combinations of settings used (all the instances in the tables in appendix C)(on the left); and the number of occurrences of each of the setting values in the top 20 algorithms with the highest F1-score (on the right)

versions of each algorithm.

From table 5.1, we observe that, surprisingly, 14 (a 2-week baseline) is the most common value. As corroborated by figure 5.3, training with 2 weeks appears to be enough, for most of the cases, and increasing the baselining period to 30 or even 60 days often results in marginal improvements or even worse F1-scores.

The Autoencoder and PCA results show some similarities, such as no significant differences in the maximum F1-score, regardless of the number of days used to baseline. However, the median F1-score obtained with PCA using a 14-day baseline is significantly higher than the 2 other settings.

In the Mahalanobis method and in the isolation forest, a 14-day baseline results in significantly higher F1-scores than the other 2 settings. Also, in the Mahalanobis method, the best score obtained using a 30-day baseline is still significantly lower than both the best score using 14 and 60 days, showing that, in this algorithm, there are multiple factors that, along with the baselining period, influence the F1-scores, as expected.

The HBOS is the only algorithm that contradicts the supremacy of the 14-day setting, with the maximum F1-score being observed with a 30-day baseline period, and the highest median score being observed with 60-day baselines.

In conclusion, a **14-day baseline** is not only sufficient to train most of the algorithms, as it **outperforms the other 2 possible settings** (1 month and 2 months), in several scenarios. Even though the best setting may be dependent on the algorithm applied, the 14-day setting appears to be highly consistent across most of the ones we experimented on.

Using 30 days of historical data to train the models also seems to be a strong choice across most of the algorithms, with the exception of the Mahalanobis method. On the other hand, the longest period experimented (60 days) is probably unnecessary as it often leads to worse results, requiring significantly more data and time needed to wait until we can start training the models, if the client is new.

In subsection 5.1.2 we analyse, from a time perspective, the difference in training time

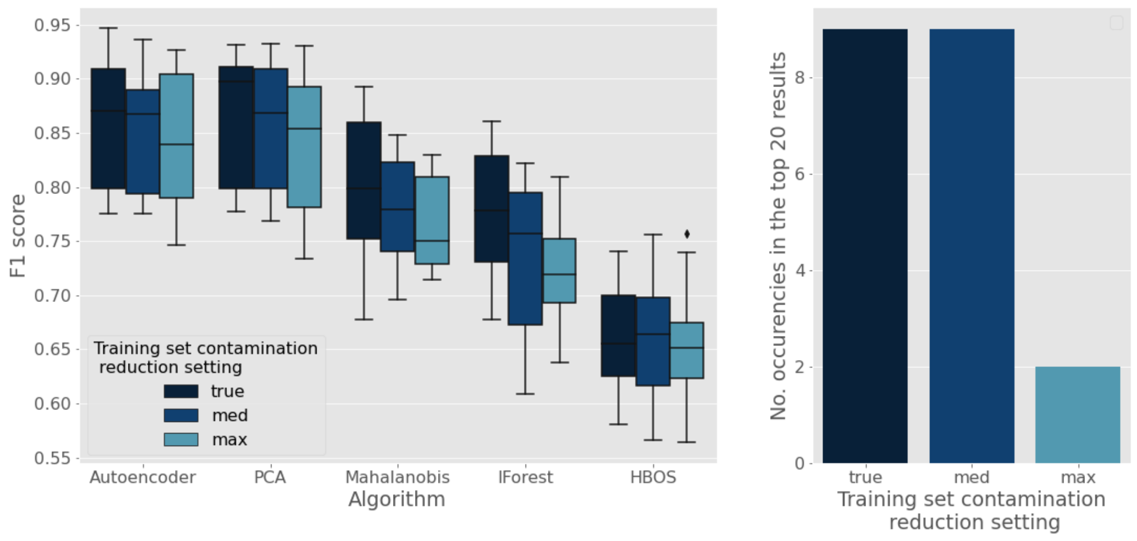


Figure 5.4: F1-score distribution for each algorithm and each value of the contamination reduction setting, for all the different combinations of settings used (all the instances in the tables in appendix C)(on the left); and the number of occurrences of each of the setting values in the top 20 algorithms with the highest F1-score (on the right)

when using the different baselining period settings.

Training contamination reduction impact

Contamination reduction is the procedure defined to remove some of the contamination present in the training data, and is a strategy used in other UEBA approaches, such as [20]. The way we attempt to achieve this is by running a robust version of PCA (the one we also use for the actual task of anomaly detection) on the training data (similarly to what the authors of the mentioned article do), and exclude the instances with the highest anomaly scores, based on some threshold given by a percentile of the score distribution. PCA was the best candidate for the application of this strategy due to its capability of dealing with training contamination, and also to the scalability shown (fastest algorithm both in training and in testing (table 5.2 and figure 5.8).

We defined three settings for this parameter: "true", which corresponds to not employing the outlier removal approach (or, in other words, use the "true" contamination present in the dataset); "med", which corresponds to removing from the training data the instances with the 1% highest anomaly scores, and "max", which uses the highest 5% as the threshold for removal. This parameter is present in the column "Training contam." in table 5.1, as well as in the tables present in Appendix C, under the same column name.

From the histogram in figure 5.4 and table 5.1, we observe that both the "true" and the "med" setting appear frequently, and the "max" setting is only seen twice, and far from the top of the table.

The box plots on the left of figure 5.4 show the distribution of the F1-score, for each algorithm, when the different contamination reduction settings are used. From these graphics, it is possible to observe that the HBOS is the only algorithm on which the "true" setting does not yield the highest F1-scores.

In general, the results seem to favor the setting in which no contamination reduction is performed ("true" setting), and the "med" setting is also better than the most aggressive,

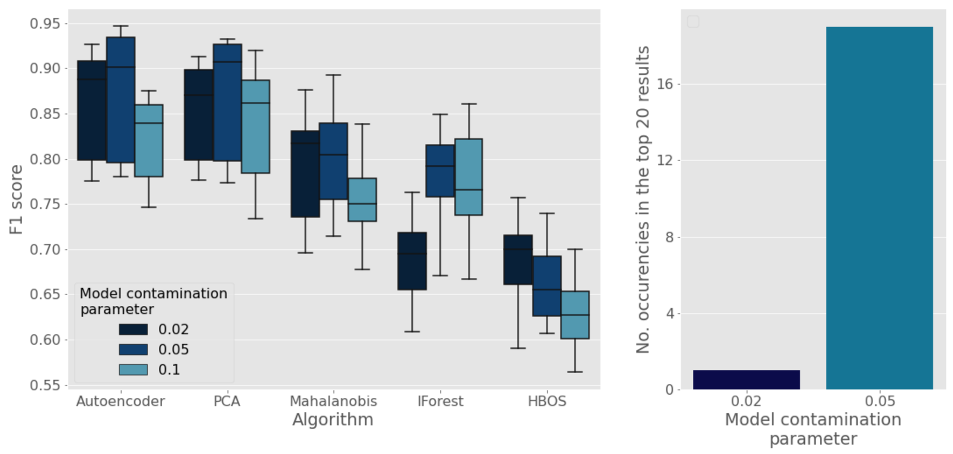


Figure 5.5: F1-score distribution for each algorithm and each value of the contamination parameter, for all the different combinations of settings used (all the instances in the tables in appendix C)(on the left); and the number of occurrences of each of the setting values in the top 20 algorithms with the highest F1-score (on the right)

"max" setting. In the autoencoder, the difference between the highest F1-score obtained with the "true" and "med" is more significant than in PCA, with the latter showing marginal differences in the top scores of each contamination reduction setting, but with a median F1-score that clearly favors the "true" setting.

In the Mahalanobis method and the isolation forest, using the "med" setting results in a significant decrease of both the maximum and median F1-score, when compared to the "true" setting, and the "max" approach is even worse.

The HBOS's results, on the contrary of all the other algorithms, obtain higher F1-scores (both maximum and median) with the "med" setting, and the highest result with the "max" setting is also higher than with no contamination reduction ("true"), showing once again how the results of this algorithm diverge from the others.

In conclusion, the approach for reducing training data contamination with a robust algorithm, is **shown not to be beneficial** in this scenario, with the evaluation scores decreasing when the strategy is applied, most of the time. Therefore, the default setting will be the "true" setting, but we will include the remaining possibilities in production deployments, as we cannot be sure that these results will be observed for all the clients.

Model contamination impact

Contamination is one of the most relevant model parameters, and is required across all the 5 models analysed in this work. When deploying the framework using data from the different clients, we will not be able to estimate the training contamination, reason why it is important to establish a default value that will not result in be balanced when it comes to the false positive versus false negative rate.

We defined three values to experiment with: 0.02, 0.05 and 0.1 (1%, 2% and 10% of training contamination, respectively). Similarly to the previous parameters, we analyse the impact of varying the parameter with table 5.1 and figure 5.5.

From table 5.1, it is possible to observe that the 0.05 setting is by far the most frequent,

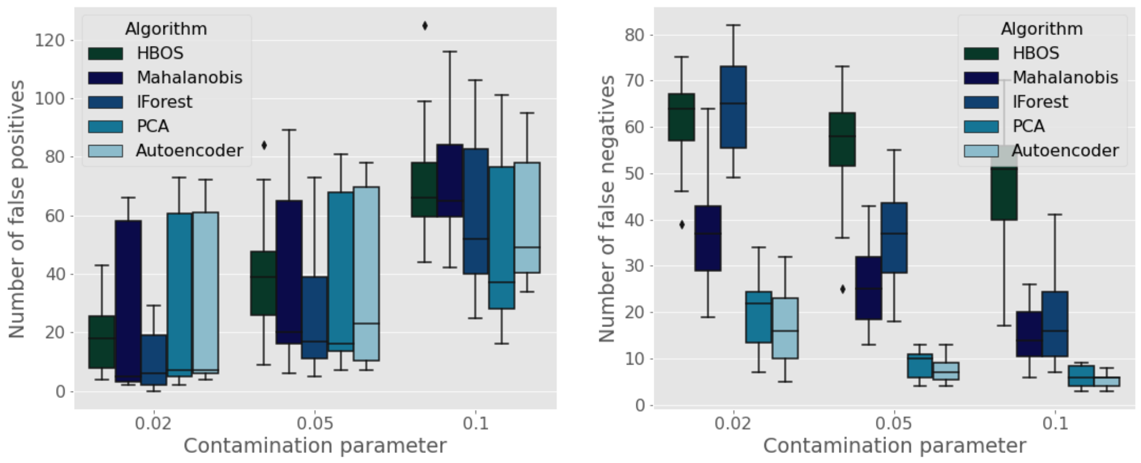


Figure 5.6: Distribution of the number of false positives (on the left) and the number of false negatives (on the right) in each algorithm and each value of the contamination parameter

with 13 occurrences in the top 4 versions of each algorithm. On the other hand, the lowest contamination value (0.02) appears five times in the table, and the 0.1 setting only 2 times, far from the top results.

The histogram on the right of figure 5.5, which shows the setting used in the 20 setting combinations which obtained the highest F1-score, regardless of the algorithm, contains 19 occurrences of the value 0.05, 1 occurrence of the value 0.02 and 0 of the value 0.1, once again showing that 0.05 is probably the most appropriate value for our dataset.

The plot on the left of figure 5.5 shows the F1-score distribution for each contamination setting, for each algorithm. The different box plots show that the 0.05 setting results in the highest F1-scores in 3 out of the 5 algorithms. In the isolation forest, however, it is with a 0.1 contamination setting that the highest F1-score is achieved, even though the median score is higher with 0.05. Also, when HBOS is used, the highest scores are obtained with a 0.02 contamination, contradicting what was observed in the other algorithms.

Even though the autoencoder is capable of achieving F1-scores of more than 0.9 with a contamination parameter of 0.02, the same does not happen with 0.1. The median score is also significantly lower in the latter setting. With PCA, however, the difference is not as substantial, with a 0.1 contamination resulting in the highest F1-score of about 0.92, and a median score marginally lower than the median score when 0.02 is used.

The contamination parameter seems to also affect the isolation forest quite significantly since, with the value 0.02, the algorithm achieves at most about 0.76 F1-score. Also, IForest is the only algorithm in which 0.1 results in a higher top score than 0.05, even though the median score is lower.

In figure 5.6, we analyse how each contamination value affects the number of false positives and false negatives, across all the algorithms, to draw the expected conclusions: independently of the algorithm, the number of false positives raises as we raise the contamination value specified (on the left of the figure) and, on the opposite direction, the number of false negatives decreases (on the right of the figure). The plots show the importance of investing in an appropriate estimation and further adjustments of the parameter, as an inaccurate value may lead to several irrelevant alarms being created, or to important cases not being reported at all.

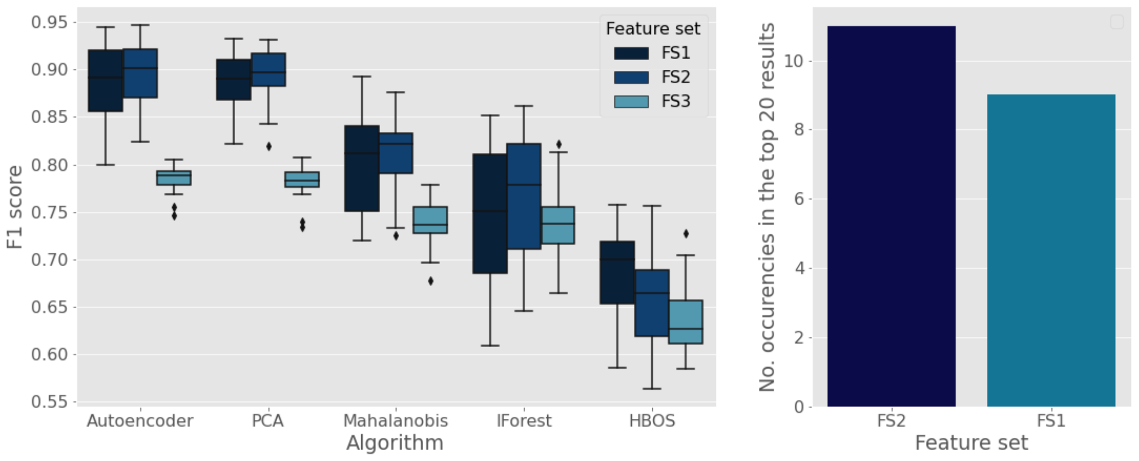


Figure 5.7: F1-score distribution for each algorithm and each feature set, for all the different combinations of settings used (all the instances in the tables in appendix C)(on the left); and the number of occurrences of each of the setting values in the top 20 algorithms with the highest F1-score (on the right)

In conclusion, the analysis conducted shows that the contamination parameter has a very strong impact on the results produced. Also, the results obtained suggest 0.05 (5%) as the most appropriate value for a default contamination setting, based on the dataset analysed. Also, this middle parameter is more conservative than 0.1 regarding the quantity of alerts that will be generated, reducing the number of false positives, but also higher than the minimum used in this experiment of 0.02, which could result in some threats being missed by the algorithm (false negatives), as shown.

The value 0.05 results in the highest F1-scores for the most promising algorithms, such as the autoencoder and PCA. However, the results also show that the most appropriate estimation of the **contamination value depends on the algorithm**, and not only on the actual outliers in the training data.

Finally, it is important to remember that we can **tune the contamination parameter** in subsequent training iterations, through the **feedback mechanism** implemented, explained in section 5.2.1.

Feature set impact

Due to the unsupervised nature of our problem, selecting the best set of features was a challenge from the beginning of the project. With that in mind, we defined three different feature sets, specified in table 4.2. With the goal of choosing the best between the three, we can identify which are the predominant sets present in the top results. From table 5.1, as well as from the histogram on figure 5.7, we are able to identify that both FS1 and FS2, the smallest and the medium-sized sets stand out, with FS3 showing up zero times both in the table and in the histogram.

From the left plot of figure 5.7, which shows the F1-score distribution for each model and each feature set, it is also clear that FS3, composed by the features in FS2 with 2 additional engineered features, is the one with the worst performance, across every single model. Even the algorithms with the best scores, in general (Autoencoder and PCA), are barely capable of surpassing an F1-score of about 0.8 with this feature set.

The bad performance seen in models using FS3 is quite unexpected, and, from table 5.1

and appendix C, it is possible to observe that the poor performance is due to a significant increase of false positives, most properly introduced by these extra features. This shows that the ratios introduced in the engineered features (ratio between the number of calls and access to PII and the ratio between the session duration and the total cost) have a high negative impact on the models' performance, and even though the correlation between the features used in the ratios are high, the violation of that correlation does not seem to help in reducing the false negatives, but instead, in most cases, raises the number of false positives dramatically. We can also draw some immediate conclusions on **how important the feature set chosen** is, with our beliefs on what would be good candidate engineered features, result in much worse performances than the feature sets composed of only raw features.

The hard decision is in choosing between FS1 and FS2. Even though the highest F1-score of 3 models (PCA, Mahalanobis and HBOS) is achieved with FS1, FS2 yields a better median F1-score across 4 out of the 5 models. The only exception is HBOS which, even with a similar best F1-score between FS1 and FS2, the median score with FS1 is significantly higher (about 0.04 points).

In general, the added features in FS2 do not seem to increase the number of false positives, showing that the features that, during exploratory data analysis (EDA), were not considered the most important, are in fact auxiliary to obtain better results, in most cases.

In conclusion, **FS2** will be used as the default feature set, to deploy for other clients, especially if scalability is not a concern when the number of features grows, which we analyse in section 5.1.2. However, FS1, with 3 fewer features, also achieves high results across the different algorithms, and could be considered in specific situations and future tests.

Execution times

Another important step of assessing the applicability of this framework into Talkdesk's production environment, which helps to decide between all the settings analysed before, as well as the algorithm itself, is the performance (regarding time), during training and test.

Even though we do not expect that training and prediction time will be a concern in the first iterations of the framework, whose algorithms will be retrained every week or every 15 days, and predictions will be made only once a day, these settings could be changed and the goal is to prepare the framework to be suitable for different time granularity possibilities (*e.g.*, 1-hour batches or even-near real-time (streaming) with moving windows). Therefore, prediction time could be a significant concern in the future, and that is the reason why it is important to compare the different algorithms in that regard.

The setup we built to evaluate the time performance of each algorithm consisted of the following steps, ran using an i3.xlarge AWS instance (4 vCPUs, 30.5GiB RAM), with the same random seed across experiments:

1. Define a baseline set of parameters to use across all the models. The chosen parameters were: the use of the smallest feature set (FS1), no contamination reduction is applied ("true" configuration), a 30-day baseline, and a 0.05 contamination (model parameter);

2. Train every model with the configurations above, for 100 times, and calculate the average and the standard deviation, for each model;

Table 5.2: Mean and standard deviation train times from 100 runs of each model. The second and third columns represent the "base" model (Feature set FS2, no contamination reduction ("true" setting) and 30 days baseline time), and the subsequent columns (on the right) contain the results of one single parameter modification, specified in the column name. All the results are presented in seconds.

Algorithm	**Avg. train time (s)**	**Std. train time (s)**	**Avg. time (FS3 (s))**	**Std. time (FS3 (s))**	**Avg. time ("med" (s))**	**Std. time ("med" (s))**	**Avg. time (60 days (s))**	**Std. time (60 days (s))**
HBOS	0.00975	0.00146	0.01089	0.00178	0.01512	0.00211	0.01108	0.00171
Mahalanobis	0.34097	0.02295	0.34357	0.02552	0.34429	0.01906	0.37437	0.01103
IForest	0.23825	0.02257	0.23443	0.01946	0.23868	0.01968	0.24022	0.02551
PCA	0.00820	0.00107	0.00898	0.00160	0.01317	0.00166	0.00918	0.00144
Autoencoder	1.67781	0.17669	1.71583	0.14453	1.84081	0.23758	2.25967	0.13440

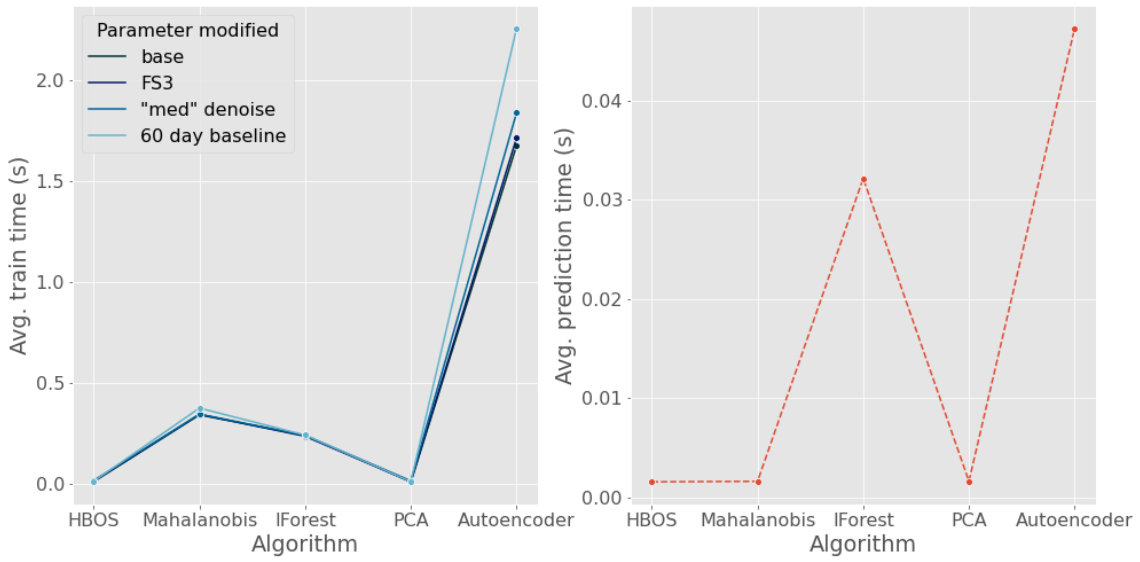


Figure 5.8: Average train (on the left) and prediction (on the right) time from 100 runs of each model.

3. Vary the parameters defined, one by one (change the feature set to the one with the highest number of features (FS3), use a 60-day baseline to train, and the "med" contamination reduction setting), and evaluate how each parameter modification affects the training time, also by training for 100 times and calculating the mean and standard deviation;

4. Evaluate the testing time performance on the base settings defined, for which we used the full data available (about 5000 rows) as a batch to predict on, and predicted 100 times for each algorithm, averaging the results.

The results related to the training part of this analysis are presented in table 5.2, and are summarized in the plot on the left of figure 5.8. On the right of the same image, we see the results of the analysis during testing/prediction time.

When it comes to train times, there are two algorithms that stand out: HBOS and PCA.

As expected, HBOS is quite efficient, due to its simplicity, with PCA being even faster, independently of the settings used. PCA also surprised by being about 40 times faster than the Mahalanobis method, and more than 200 times faster than the autoencoder.

From the plot on the left of figure 5.8, we can also observe that the setting with the highest impact on training time was changing from a 30 to a 60-day baseline, which expectably doubles the amount of training instances. In the autoencoder the difference is quite significant, raising the time required for training from 1.67 seconds to 2.25.

The contamination reduction component, which requires training PCA to remove (in the case of the "med" setting) 1% of the contamination, and using the feature set FS3, instead of the base FS1, are changes that result in marginal train time differences, in most of the cases. The results show that the contamination reduction algorithm is efficient, and the time consumed in training PCA is most properly compensated (in the final training time) by the number of instances removed. Also, a certain increase in the number of dimensions of training does not affect the training performance of these particular algorithms, especially when compared to an increase of the number of instances.

The performance in testing time is currently more important for our use case, especially if a near-real-time scenario is applied. The experiments were conducted with a simulation of a batch of about 5000 instances, and the results, shown in the right plot of figure 5.8, show that the prediction times are acceptable for a production scenario, in all the 5 models. Even though the plot shows a significant difference between the autoencoder and the other algorithms, we can observe that it takes, on average, a bit more than 47ms to process the mentioned batch. Nevertheless, it is important to highlight that PCA is the fastest algorithm also at test time, taking on average 1.6ms to compute the entire batch.

Key takeaways

In this section, we used a labeled test set to compare the different algorithms, as well as settings such as the baselining period, the feature set and the contamination reduction in the training data. The autoencoder achieved the highest F1-scores in the test set, showing that the neural network approach is indeed the most powerful, in our scenario. Nevertheless, the scores obtained with PCA are quite interesting as well, clearly standing from the remaining algorithms.

The results also show that, in general, 14 days of training data is enough to achieve similar or even higher F1-scores than the other baselining period settings, meaning that, when a new client is added, there is a strong belief that we can start detecting anomalous behavior after this period. Also, we concluded that the contamination parameter is dependent both on the dataset and the algorithm, making it very hard to estimate the most appropriate value for every client, in an automated fashion. Hopefully, the parameter can be adjusted with the help of a feedback mechanism, introduced in 5.2.1.

We have also concluded that the feature set applied is determinant for the success of the algorithms. The engineered features resulting from highlighting correlated pairs of features (used in FS3) reveal significant worse results, reason to favor the raw features present in FS1 and FS2.

With consistent results (the closest to the autoencoder's) and a proven performance in train and test time, **PCA is the overall best performer of this analysis**, making it the most suitable algorithm for the deployment step, along with a 14-day baseline period, the "true" setting corresponding to no contamination reduction, a 0.05 value for the contamination parameter and, finally, the middle-sized feature set (FS2).

Table 5.3: The 4 settings of each algorithm with the highest F1-score, using data from supervisors as the positive class, sorted by decreasing order of the metric.

Algorithm	Training set parameters			Model parameters	Metrics					
	Baseline period	Feature set	Training contam.	Contam.	F1	ROC	Prec.	Rec.	FP	FN
IForest	14	FS3	true	0.02	0.945	0.958	0.964	0.926	6	13
Mahalanobis	14	FS2	true	0.02	0.944	0.956	0.97	0.92	5	14
PCA	14	FS1	true	0.02	0.944	0.954	0.976	0.914	4	15
Mahalanobis	30	FS1	true	0.02	0.942	0.956	0.964	0.92	6	14
Mahalanobis	14	FS1	true	0.05	0.942	0.956	0.964	0.92	6	14
PCA	30	FS1	true	0.02	0.941	0.954	0.97	0.914	5	15
Mahalanobis	14	FS1	true	0.02	0.941	0.954	0.97	0.914	5	15
PCA	60	FS1	true	0.02	0.941	0.954	0.97	0.914	5	15
Autoencoder	14	FS2	true	0.05	0.939	0.959	0.948	0.931	9	12
Autoencoder	14	FS2	true	0.02	0.938	0.953	0.964	0.914	6	15
Autoencoder	60	FS1	true	0.02	0.938	0.953	0.964	0.914	6	15
Autoencoder	14	FS1	true	0.02	0.938	0.953	0.964	0.914	6	15
PCA	14	FS2	true	0.02	0.938	0.953	0.964	0.914	6	15
IForest	14	FS2	true	0.05	0.936	0.952	0.958	0.914	7	15
IForest	14	FS2	true	0.02	0.929	0.942	0.969	0.891	5	19
IForest	14	FS1	true	0.05	0.929	0.946	0.958	0.903	7	17
HBOS	30	FS3	true	0.02	0.86	0.917	0.845	0.874	28	22
HBOS	14	FS3	max	0.02	0.857	0.91	0.857	0.857	25	25
HBOS	60	FS3	max	0.02	0.85	0.914	0.827	0.874	32	22
HBOS	14	FS3	true	0.02	0.848	0.904	0.851	0.846	26	27

5.1.3 Sampling from supervisor activity

In [43], the authors take inspiration in an approach called negative sampling, used in fields such as natural language processing (NLP) and computer vision, to evaluate their unsupervised UEBA solution. The authors employ this strategy when modeling single users, and start by assuming the test data put aside for testing each user's model is free of anomalies (thus belonging to the negative class), and replace some of that data with the **daily activity of other random users**, labeled with the positive class. The approach allows evaluating whether each model can properly distinguish the activity of a user or if it fails to do so.

In section 3.4 we have identified differences between the behavioral vectors collected for users, supervisors and administrators, confirming the assumption that different roles in the contact center behave differently, within our set of features. Thus, evaluating whether the trained models can properly distinguish agents from supervisors or administrators can be an **automated complementary effort** to ensure the baselines established are solid and effective.

In our dataset, about 10% of the data ($\approx$ 500 rows) correspond to supervisor and administrator vectors. To do this procedure, we add to our testing set (labeled with the negative class), corresponding to about 30% of the total data, a sample of 150 instances from the supervisor/administrator behavior vectors (labeled with the positive/anomalous class), to preserve the ratio between classes used in subsection 5.1.2.

We conducted this experiment on the top-performing settings (highest F1-score) of each algorithm, present in table 5.1, producing the results shown in table 5.3.

Similarly to what was done in section 5.1.2, table 5.3 shows the settings used in each of the algorithms, as well as several metrics (F1, precision, recall, AUC-ROC, the number of

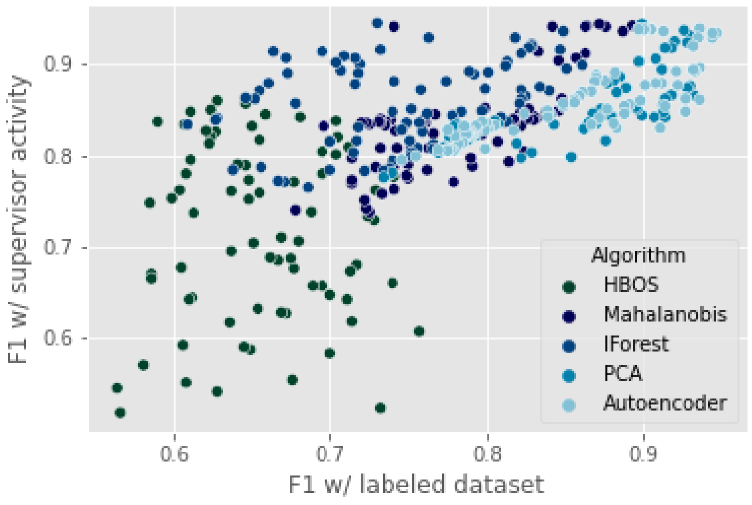


Figure 5.9: Scatter plot between the F1-scores obtained with the labeled dataset and the sampling from supervisor activity

false positives and false negatives).

From table 5.3, we can conclude that most of the settings get similar scores and a high rate of success in distinguishing between agents and superior roles. We were able to confirm that the false negatives are related to days with an absence of significant activity, (*e.g.*, a session, with none or few events), which makes it hard to distinguish between the roles, with the feature sets employed. The false positives are also explained by days with less activity from agents, but with some events occurring more spread throughout the day, leading to bigger sessions, which is more typically seen in supervisors' behavior.

Even though it is the isolation forest (IForest) that achieves the highest F1-score in this experiment, the advantage over Mahalanobis and PCA is negligible as they are behind by only 0.001 points.

The biggest curiosity is observed with the presence of FS3 in the best result, which we observed to worsen the results significantly in the evaluation with labeled data. However, it only appears in the top isolation forest score and the HBOS's scores, with both the remaining feature sets also being present in the selected algorithm from the previous analysis (PCA), with high scores.

Even though this approach is **not conclusive on determining the algorithm's capability of detecting relevant security threats**, it has the advantage of not requiring ground truth labels, making it a possible easily automated evaluation strategy for the production deployments, which should be capable of reporting whether the baselines can accurately capture the behavior of agents.

Also, we were able to identify a significant positive correlation between the results of the labeled activity and those of this approach, as demonstrated in figure 5.9. The scatter plot shows that especially in the algorithms with higher F1-scores, such as the autoencoder, PCA and the Mahalanobis method, this correlation appears to be stronger, which shows that this is, indeed, an **helpful automated way of assessing the models deployed**.

Finally, we can also, in the future, compare the results of the strategy applied throughout this section with the results obtained from the feedback mechanism, which we present in

Table 5.4: Evaluation of the interpretability methods

Algorithm	Score top N	Score top N+1	Score top N+2	Batch time (s)
z-score relative importance	0.817	**0.898**	**0.944**	**0.01**
DIFFI	0.687	0.753	0.844	3.33
Isolation forest SHAP (TreeExplainer)	0.739	0.781	0.798	0.36
Mahalanobis SHAP (KernelExplainer)	0.522	0.611	0.667	7.46
PCA SHAP (KernelExplainer)	0.815	0.885	0.922	7.58
Autoencoder SHAP (KernelExplainer)	**0.822**	0.891	0.930	30.8

section 5.2.1.

5.1.4 Evaluation of the interpretability methods

In this section, we evaluate the interpretability methods used along with each algorithm to explain the anomalies outputted. When creating labeled instances of threats to evaluate the algorithms and other settings (section 5.1, we also created labels containing the feature or set of features that contributed to each anomaly.

The strategy used in this section consists of evaluating three interpretability algorithms, presented in section 4.2 (z-score, depth-base feature importance for the isolation forest (DIFFI) and Shapley additive explanation (SHAP)) using our labeled anomalies, ranking the features by the metric of importance yielded by the algorithms in decreasing fashion, and comparing to the top N rated features (the most important according to the algorithm) to the ground truth features.

Our metric consists of the average percentage of the N ground-truth features present in the top-ranked features. Therefore, if the ground truth consists of only one feature, the score for that instance will either be 1 if that feature is ranked in the first place by the algorithm, or 0 if any other feature places first. If the ground truth consists of two features, the score will be 1 if the algorithm places both the features at the top, 0.5 if it places only one of them at the top, and 0 if none are present. The final score for an algorithm consists of the average of the score obtained for each of the 150 instances.

Table 5.4 contains the average score obtained when varying the size of the ranked list of features. The "score top N" column corresponds to the mentioned score metric when the ground truth is compared to the N top-ranked features by the algorithm, and the columns to the right correspond to increasing this N by 1 and 2. The "Batch time" was measured by running the 150 instances through each algorithm 100 times on the same i3.xlarge machine used in subsection 5.1.2, and averaging the results. Both DIFFI and SHAP require a trained instance of each algorithm to be provided, and we decided to use the combination of settings with the highest F1-score (present in table 5.1).

The results obtained are quite surprising, in the sense that the most simple approach (z-score) is the one that obtains the best score at top N + 1 and top N + 2, also having the lowest time to provide the interpretations.

In the second row we have DIFFI, and the model-specific approach also surprised by the low score when compared to some of the KernelSHAP tests. However, the TreeSHAP on isolation forest also shows a poor performance, even with a bit higher result than DIFFI on top N and N + 1, performing worse on top N + 2. This could be explained by the worse performance of the isolation forest in our UEBA scenario when comparing to the PCA and the autoencoder. Finally, DIFFI is about 10x slower to process our batch of about 150 instances than the TreeExplainer.

When we use KernelSHAP to interpret the Mahalanobis method's outputs, the results are the worst across all the experiments, when it comes to the scores obtained. The score at top N is about 0.52, which is about 0.3 points lower than the most simple approach, z-score. Also, the highest score with the Mahalanobis version is about 0.28 points lower than the best-performing approach. We were not able to find a reason to justify the discrepancy of scores between the Mahalanobis SHAP and the PCA SHAP.

The best results using SHAP's KernelExplainer were obtained for PCA and the Autoencoder, which are also the better performing algorithms in the anomaly detection evaluation. Explaining the anomalies with the autoencoder and KernelSHAP gives the best results at top N (0.822), and it results in a bit of improvement comparing to the results with PCA + KernelSHAP. The biggest concern with both these methods is related to the time taken to process the batch, especially with the autoencoder (about 30s, 100x more than z-score).

With the results observed, we have no significant doubts that **the z-score approach**, employed in [20], **is the one we will preferably use**. However, this is a problem in which we intend to invest more time in the future.

5.2 Deployment

The last stage of the validation consisted of deploying the framework in a staging environment, using real data from two distinct clients with different characteristics, monitoring the results for a period of two weeks. We included in the deployment a feedback mechanism (in which we act as the analyst in the client), which we present at the beginning of the section.

5.2.1 Feedback mechanism

With the absence of ground truth labels to help validating the models for multiple clients, we expect feedback to be a central way to monitor and evaluate how effective the framework is.

The feedback mechanism consists of introducing the analyst on the customer side in the loop, providing information on whether each specific anomaly is relevant or not. Figure 5.10 shows how this is accomplished.

In the first iteration, the only possibility for feedback will be: positive, not sure, and negative, as we assume the simpler the feedback the more chances we have of the analyst participating in the initiative. Positive feedback implies the anomaly is a true positive which was easily interpreted by the analyst. A neutral answer indicates the analyst was not

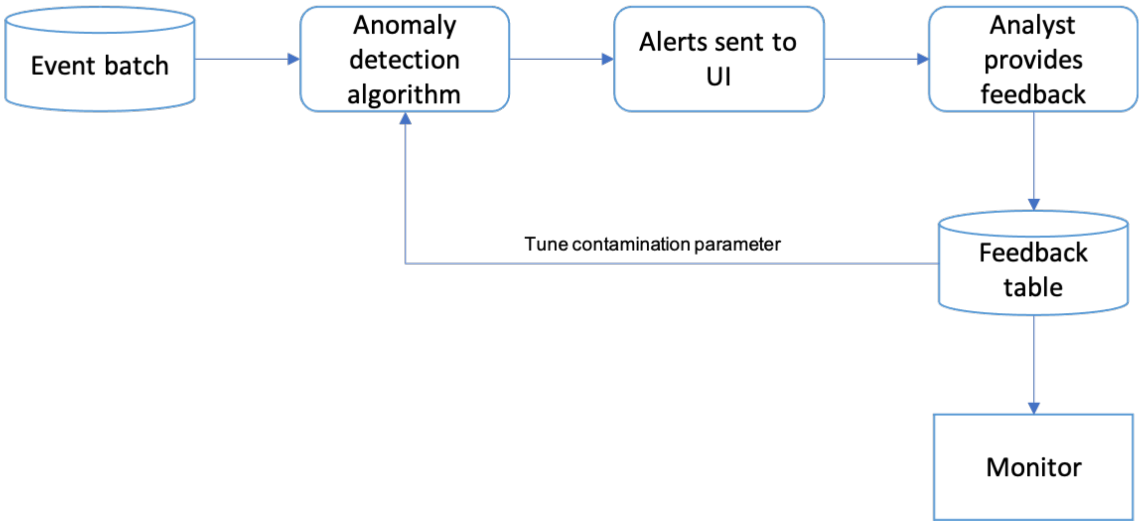

Event batch
Anomaly detection algorithm
Alerts sent to UI
Analyst provides feedback
Feedback table
Tune contamination parameter
Monitor

Figure 5.10: Feedback mechanism loop

capable of assessing whether the alarm was in fact something relevant, possibly indicating issues in the interpretability methods, and a negative response indicates a false positive.

In this scenario, the provided feedback is stored along with the anomaly identifier in the Feedback table.

The anomalies reported as true positives by the client, corresponding to correctly reported cases, **are not used to in the subsequent training iterations of the model**, hopefully leading to a successive contamination reduction over time. On the other hand, the anomalies with negative feedback, corresponding to false positives, are included in training instances, even when they surpass the baselining period (*e.g.*, if an anomaly was reported as a false positive 2 months ago, and the baseline period is 14 days, the instance will still be included in the training set).

In the future, we also expect to use the feedback to **create and explore labeled datasets**, which we could use to turn the problem into a supervised one.

Finally, the feedback can also be used for **reporting** purposes, as well as to **tune the contamination** parameter of the anomaly detection algorithm in subsequent training phases.

Reporting feedback

The feedback table will be used to monitor each account's ratio between positive, neutral, and negative feedback. To do so, we will calculate this ratio periodically (*e.g.*, once a day) and define thresholds that, when exceeded, we should be notified. Some immediate actions on excessive neutral or negative feedback include:

- Analyse the anomalies with neutral and negative feedback for the account. Evaluate manually each anomaly to identify potential issues, either with the interpretability method (more important in the case of neutral feedback) or the anomalous record itself (try to understand why the algorithm made the decision);
- Stop the model, retrain and used the feedback as labels to test and tune the model;
- Request more detailed feedback and adjust the interpretability ranking.

Hyper-parameter tuning based on feedback

The feedback obtained from customers will also allow readjusting the models' parameters, namely the training **contamination**. As explained in previous sections, contamination is a parameter present in all the algorithms assessed, and its influence was found to be very significant. Also, different training sets used for the various clients will contain different contamination rates, which are hard to estimate without manual effort.

Due to the need of automatic deployment of the models, we start with the same default contamination parameter for every client (defined empirically as 5%), and that is when feedback becomes strongly relevant, as it allows to readjust the parameter in subsequent training activities, having in account the feedback provided, in one of two ways:

- if any false positives are registered, we reduce the contamination parameter by the percentage of the negative feedback received, as shown in equation 5.1, where ε corresponds to the percentage of false positives reported by the client, during the period t which, by default, corresponds to one week, a period after which the models are retrained. The contamination for the following week c_t is thus given by the contamination of the previous week multiplied by one minus ε;
- if no false positives are reported, we increase the contamination by 1%.

$$c_t = (1 - \varepsilon)\, c_{t-1} \tag{5.1}$$

This approach, which relies totally on the contamination parameter of the model deployed, has some drawbacks, such as the lack of guarantee that reducing the contamination will address the issue of a specific false positive. One different approach, used in [20], consists in assigning weights to each of the features, and use the interpretability methods, which identify the features responsible for a determined anomaly, to decrease the weight of features contributing to the false positives reported. This is an approach we intend to explore in the future, and is out of the scope of this work due to the sensitivity of the feature weighting topic.

A specific example to compare the two approaches would be a client that is not interested in the number of sessions a user creates. If several anomalies on the number of sessions are reported, the analysts in the client will send negative feedback every time, and reducing the contamination may not help in this case, especially if those anomalies have high scores. On the other hand, reducing the importance of the features related to the sessions created is more likely to help significantly.

5.2.2 Staging deployment

The last empirical validation step consists of assessing how well the framework generalizes to different Talkdesk clients. To do so, we implemented and deployed it in a staging (STG) environment, and trained it independently for 2 different clients, to which we refer in this document as client A and client B, monitoring and evaluating the results for two consecutive weeks.

The implementation of the ETL process, as well as the adaption from the exploratory code to make it ready for deployment **was highly time-consuming**, contrarily to what

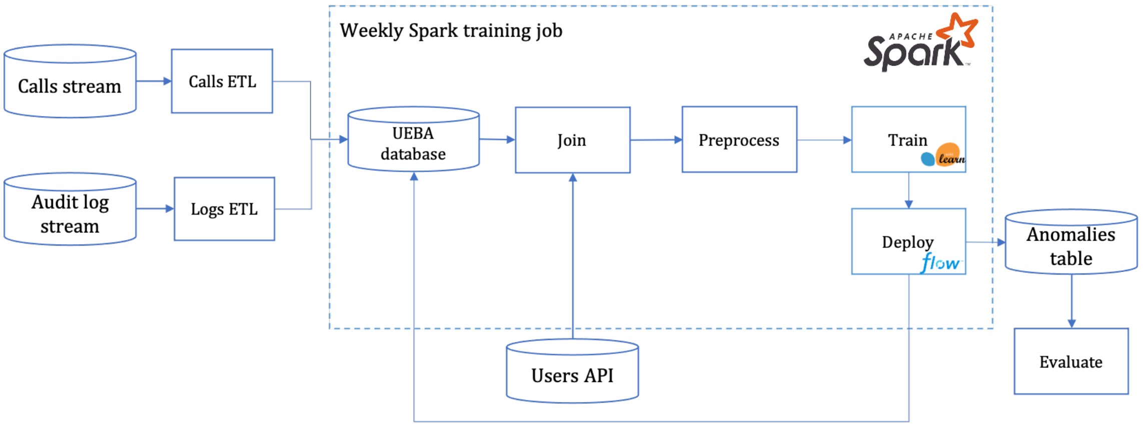


Figure 5.11: Model deployment and evaluation process

we expected when planning the work. Nevertheless, we believe that having results of a deployment, even if it is not in production, enriches the analysis of the results.

In this section, we present and analyse the metrics related to the anomalies detected.

Figure 5.11 shows the end-to-end process developed for this deployment. It starts with the consumption of the two streams containing call and audit logs data, which are sinked into separate tables in our relational database. When the Spark training job is triggered, the latest 14 days of data for each client are retrieved, processed and merged following a pipeline similar to the one presented in section 3.3, a process which results in a spark dataframe with each row corresponding to the activity of a user in a determined day, ready for training.

The data is fitted on pyOD's PCA, selected as the most promising algorithm for deployment, which uses the implementation of scikit-learn. We use MLflow to help with deployment and monitoring, and sink the anomalies into a relational table, for further evaluation. We used the default settings defined empirically in section 5.1, namely a 14-day baseline, no contamination reduction (the "true" setting), a 0.05 (5%) contamination parameter, and the feature set FS2.

We were able to do this experiment over a 30 day period, which we divide into 4 weeks, meaning 2 cycles of training (with a 2-week baseline), and 2 weeks of evaluation: in the first cycle, PCA is fitted with the data from week 1 and 2, using the default parameters, and evaluated on week 3; in the second cycle, the fit operation happens on the data from week 2 and 3, with an adjustment of the contamination parameter, based on the feedback mechanism, and the anomalies from week 4 are registered and evaluated.

Results for client A

The first client used for deployment purposes, referred to as client A, is composed of about 100 users, similarly to the data used in the analysis throughout the document. However, the contact center of client A is open at the weekends, and the working hours are radically different compared to those of our first dataset, mainly due to timezone differences.

The results for client A, after following the mentioned procedure for training, deployment, and evaluation are present in table 5.5.

From the table, we can extract the most important results from the two-week deployment, with 14 days of training in each week (corresponding to about 900 rows of data).

Table 5.5: Results of applying PCA to client A along 2 weeks

Week	**# rows train**	**contam.**	**# anomalies**	**# distinct users**	**# TP**	**# FP**	**Interp. score @ 2**
1	910	0.05	18	8	16	2	1.0
2	890	0.044	12	6	10	2	1.0

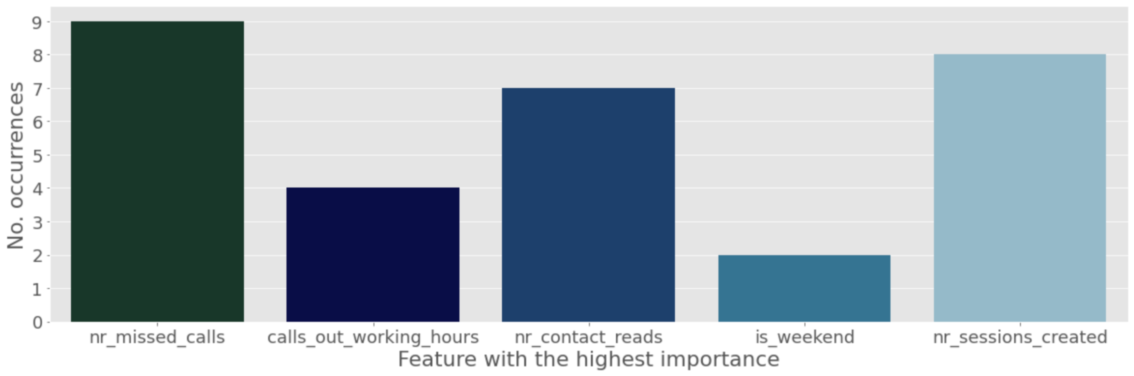


Figure 5.12: Number of times each feature was ranked as the most important in the anomalies reported throughout the 2 weeks, for client A

From the first to the second week, we notice a smaller amount of anomalies reported, either due to the contamination parameter used (0.045 instead of 0.05, due to the feedback mechanism), or due to the users incurring in less anomalous activity. Nevertheless, there was an increase in the percentage of false positives in week 2, from about 11% to 16%, which is not conclusive as we need more time (more weeks) to be able to draw more representative conclusions, mainly on the feedback approach.

The results also show a strong amount of what we believe to be true positives, after careful analysis of the anomalies. Also, the interpretability method employed, given by the z-score, obtained a score of 1.0 in pointing to the features responsible for the anomalies, when we use the 2 top-ranked features to do the explanation.

Figure 5.12 shows the distribution of the top feature for each of the 30 anomalies registered during the 2 weeks, or, in other words, the feature that the interpretability method used considered as the most important for each of the anomalies reported. The figure shows that, for client A, there is a relative balance between the features that contribute the most to the different anomalies and, in this case, the alerts tend to be related to threat scenarios such as performance decrease (caused by a high number of missed calls), data breach (caused by an unexpected number of access to personal data), and account compromise (caused by a high number of sessions, from multiple devices).

The **key takeaways** from the experiment on client A include a reasonable ratio between true positives and false positives, great results on the interpretability of alerts, and a lack of effectiveness of the feedback approach in reducing the false positives. Also, the anomalies reported may relate to some of the threat scenarios identified in table 3.3, according to our manual analysis.

Table 5.6: Results of applying PCA to client B along 2 weeks

Week	**# rows train**	**contam.**	**# anomalies**	**# distinct users**	**# TP**	**# FP**	**Interp. score @ 2**
1	1135	0.05	27	19	26	1	0.953
2	1147	0.048	19	12	15	4	0.875

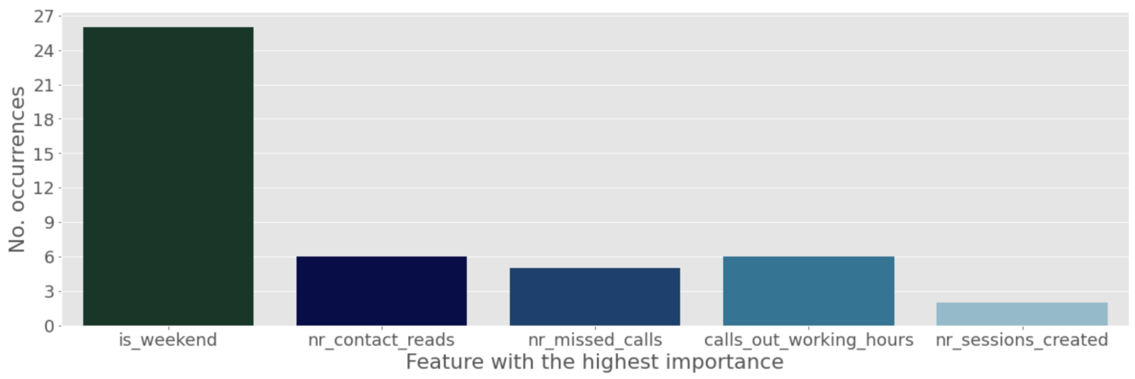


Figure 5.13: Number of times each feature was ranked as the most important in the anomalies reported throughout the 2 weeks, for client B

Results for client B

Table 5.6 contains the results of the framework deployment for client B, which is a larger client, composed of about 500 agents. Even though there are about 5 times more agents than in client A, the training data consisting of 2 weeks of activity contains about 1100 rows (only 200 more), which is probably explained by client B having fewer full-time agents, and not a significantly higher amount of concurrently active agents.

Similar to what was done for client A, we used two weeks to train PCA, corresponding to 1135 rows of behavior instances, with the default 0.05 contamination parameter. During the first week after training, 27 anomalies were observed and analysed, with us reaching the conclusion that only 1 of those was a false positive. Also, when we use the two highest scored features to explain the anomalies with the z-score method, we got a 0.953 score.

Moving to week 2, we use the two previous weeks to retrain, with a contamination of 0.048, resulting of the contamination adjustment from equation 5.1. However, we observe a significant rise in the false positive ratio, which is about 20% (4 out of 19 anomalies). Also, the interpretability score is the lowest, with the z-score approach not being capable of explaining about 12% of the true positive anomalies with the 2 highest rated features, mostly due to the anomalies being caused by feature correlations. In particular, according to our manual analysis, some of the anomalies that z-score was not able to explain are caused by data access (under the "nr_contact_reads" feature) during the weekend ("is_weekend" feature).

Similarly to the procedure with client A, we show, with figure 5.13, the feature that was ranked at the top by the interpretability method on each of the anomalies. The features that contribute the most to the anomalies reported, during the 2 weeks, for client B do not follow the tendencies observed for client A: in this case, more than half of the alerts are explained by activity during the weekends, most of the time along with accesses to PII and/or calls out of working hours.

In conclusion, for client B it would be particularly useful for us to collect real feedback from the analysts/supervisors in the client, since, even though weekend activity is rare for this client, we are not capable of being 100% sure that this is, in fact, potentially fraudulent behavior.

Current limitations

The experiment conducted and presented in this section allowed us to take some final conclusions before the expected production deployment. With this approach, we were capable of preparing the infrastructure to process the activity of Talkdesk's clients, create grouped vectors representing agents' behavior across several dimensions and, finally, use PCA to detect potential threats.

During the 2 weeks for which we conducted a manual analysis of the anomalies reported for 2 different clients, with several distinct characteristics, the results show that the framework is indeed capable of detecting relevant insights of potential threats, even with the default settings defined. Even though we did not identify any conclusive/definitive fraud practice, the algorithms are capable of identifying, with high accuracy, several signs of anomalous behavior, both at the security and business level.

Furthermore, the interpretability method used was capable of correctly identifying most of the features responsible for the anomalies, especially if the anomalies are caused by one or more univariate extreme values. However, there is a margin for improvement concerning the employed method, as we have identified **some weakness** in interpreting **anomalies caused by the correlation of two or more features** (in client B).

Although we were not able to properly test the feedback mechanism's impact on the contamination parameter, due to the need for real feedback from the analysts on the client-side, we reckon that **the approach implemented can be significantly improved**. After deploying the framework in production, we intend to analyse the results of a pre-determined amount of feedback, and identify potential strategies to act based on the insights. Also, we do not yet have a strategy to suppress subsequent anomalies similar to those reported as false positives: if a user behaves differently than the whole population continuously, and that is not seen as a problem by the client's analyst - expressed through feedback - we need to ensure that we do not send that same alert every day.

The data sources used in this experiment are only a subset of the possible interactions between users and Talkdesk's application, and do not cover, for example, activity related to permissions, associated with threat scenarios related to privilege escalation or abuse; and interaction with call recordings, which contain sensitive information (potentially PII) and should also be monitored. In the future, it is desirable to include more variate data sources and features.

Finally, despite the results for client A and client B not showing significant issues with the ratio between the true and false positives reported, we need to experiment with more clients (with different configurations and working hours) and ensure that we have this expected consistency across all of them, which is yet to be accomplished. Nevertheless, from the results obtained in this work, we have all the confidence on the value the clients will extract from the framework.

5.3 Threats to validity

In this chapter, we analysed and compared how several algorithms and settings performed on the exploratory dataset we had available for this work. Since we used this dataset to extract most of the decisions and conclusions, we must recognize that some threats to the validity exist.

In the contact-center domain, each client organization has its own specificities, working rules, dimension, location (resulting in different time zones and working cultures). Thus, it is possible that some of our conclusions may not generalize or apply to every Talkdesk's customer, reason why we need, in the future, to extend the exploratory analysis to several different clients, and identify significant divergences.

As it will not be possible to explore all the clients for which we intend to deploy the framework, we expect the feedback mechanism to be an automated way for us to monitor the quality of the framework in each individual customer.

Chapter 6

Conclusions and future work

In this work, we evaluated the applicability of a user and entity behavior analytics (UEBA) framework to use in Talkdesk, to give customers the awareness of suspicious behavior incurred by the staff, which could be related to security threats. We aimed at a multivariate scalable approach, and also capable of providing interpretable results.

With the exploratory data analysis (EDA) conducted, along with background knowledge, we were able to identify, from multiple data sources, the most relevant features to monitor user behavior. Afterwards, we selected several different anomaly detection algorithms (histogram-based outlier score (HBOS), Mahalanobis method, principal components analysis (PCA), isolation forest and autoencoder), along with 3 distinct interpretability algorithms (z-score, depth-base feature importance for the isolation forest (DIFFI) and Shapley additive explanation (SHAP)) to compare.

The most laborious step of the work consisted in the validation, where we used different strategies, some of them oriented towards future improvements, such as a feedback mechanism, from which we can collect metrics of the relevance of the anomalies reported, provided by analysts on the client-side.

We also created a labeled dataset with several examples of threats with different levels of abnormality, which we used to evaluate the different algorithms, along with different settings, including the period of historical data used to train the models, the feature set applied, and others. In this experiment, the autoencoder and PCA were the algorithms with the best results (0.97 and 0.95 ROC score, respectively), and we identified that 14 days of historical data is enough, in this dataset, to obtain such results. Also, we verified that both the feature set used and the contamination parameter (that estimates the percentage of outliers present in the training set) are critical for obtaining high classification scores.

With a method based on negative sampling, we used supervisor and administrator activity to perform an automated way to evaluate if the anomaly detection algorithms can accurately distinguish between activity vectors of different roles. In this experiment, the algorithms showed a good capacity in distinguishing between activity vectors from agents and supervisors, with the isolation forest, PCA, Mahalanobis method, and autoencoder achieving maximum ROC-scores of about 0.95.

Finally, we deployed the framework on a staging environment, using PCA along with the settings that resulted in the highest F1-score in the analysis with the labeled dataset. We used data from two different clients for a period of 1 month, and analysed the anomalies reported, as well as the results of the interpretability algorithm (z-score based). In a period of two weeks, a total of 76 anomalies were registered, incurred by 45 distinct agents, and caused by several different types of behaviors, such as a high number of accesses to personal data, activity outside of the working hours of the organization, a high number of calls missed, between others. From the 76 anomalies, 67 were identified as true positives, and the remaining 9 as false positives. The interpretability algorithm was capable of pointing to the fields that explain the anomalies reported in most cases, with the exception of cases in which the correlation of multiple fields was the cause.

Based on the results obtained, we strongly believe that the UEBA framework implemented is capable of providing significant value to Talkdesk's clients, and provide relevant alerts of potential security threats, especially since agents are, in most cases, working from home due to the pandemic of COVID-19, which is most likely to endure. After some minor improvements, mainly regarding the interpretability algorithm and the feedback mechanism, the framework will be ready for the first production deployment, aligned with the project's deadlines and activities.

This work was an introductory step of an ambitious and complex framework to monitor user behavior. Even though we were able to build most of the structure to make it ready for production, there is still plenty of work to be done, related to deployment, monitoring, and developing new capabilities. The main challenges we intend to tackle in a near future include:

- Production deployment: the sooner we have clients using the framework and providing feedback, the sooner we will be able to define more concrete actions to improve the results.
- Implement a risk-score calculation formula/strategy capable of indicating the security risk associated with each user monitored. This formula should have into account the past anomalies detected for the user as well as other indicators out of the scope of this work, such as alerts issued by a rule-based system. The risk score should accurately indicate how likely a determined user is to incur in new anomalies/security threats.
- Refactor the direct impact of the feedback on the models, as the naive strategy implemented will most likely continue to prove ineffective.
- Continue EDA activities, and add more features and threat scenarios to the current capabilities;
- Research, simulate, and evaluate the impact of attacks to the framework and, more specifically, the anomaly detection algorithms applied, such as an attacker being able to impersonate and mimic the behavior of a compromised legitimate user.
- Evaluate the applicability of supervised or semi-supervised learning, based on the feedback collected and stored.
- Explore different time granularity's for the aggregations performed on user activity (*e.g.*, 8h or 12h, instead of the employed 24h), as well as (near) real-time use-cases.
- As the amount of data grows, it is possible that the current settings and specifically PCA may not be enough. Therefore, we need to monitor for model degradation and, if necessary, invest in researching other approaches, such as the autoencoder or generative adversarial network (GAN).

References

[1] EY. Security operations centers — helping you get ahead of cybercrime, 2014. Accessed: 2020-09.

[2] Gurucul. The role of security analytics in zero trust architectures. https://gurucul.com/news/the-role-of-security-analytics-in-zero-trust-architectures, 2020. Accessed: 2020-09.

[3] HIMSS. 2019 himss cybersecurity survey. https://www.himss.org/sites/hde/files/d7/u132196/2019_HIMSS_Cybersecurity_Survey_Final_Report.pdf, 2019. Accessed: 2020-09.

[4] Gorka Sadowski, Avivah Litan, Toby Bussa, and Tricia Phillips. Market guide for user and entity behavior analytics. 2018.

[5] Chris Brook. What is user and entity behavior analytics? a definition of ueba, benefits, how it works, and more, 2020.

[6] Talkdesk. Talkdesk customer success stories. https://www.talkdesk.com/customers/. Accessed: 2020-10.

[7] CISCO. Cisco contact center global survey 2020. https://www.cisco.com/c/dam/en/us/products/collateral/contact-center/cc-global-survey-ebook.pdf, 2020. Accessed: 2020-09.

[8] Exabeam. User and entity behavior analytics. https://www.exabeam.com/siem-guide/ueba/. Accessed: 2020-10.

[9] IBM. Ibm qradar user behavior analytics. https://www.ibm.com/products/qradar-user-behavior-analytics, 2020. Accessed: 2020-09.

[10] Gurucul. Gurucul. https://gurucul.com/, 2020. Accessed: 2020-09.

[11] Christoph Molnar. *Interpretable Machine Learning.* 2019. https://christophm.github.io/interpretable-ml-book/.

[12] Varun Chandola, Arindam Banerjee, and Vipin Kumar. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3):1–58, 2009.

[13] Markus Goldstein and Seiichi Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. *PloS one*, 11(4):e0152173, 2016.

[14] Markus Goldstein and Seiichi Uchida. Behavior analysis using unsupervised anomaly detection. In *The 10th Joint Workshop on Machine Perception and Robotics (MPR 2014). Online*, 2014.

[15] Alejandro Correa Bahnsen, Djamila Aouada, Aleksandar Stojanovic, and Björn Ottersten. Feature engineering strategies for credit card fraud detection. *Expert Systems with Applications*, 51:134–142, 2016.

[16] Andrea Dal Pozzolo. Adaptive machine learning for credit card fraud detection. 2015.

[17] Markus Goldstein and Andreas Dengel. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. 09 2012.

[18] Roy De Maesschalck, Delphine Jouan-Rimbaud, and Désiré L Massart. The mahalanobis distance. *Chemometrics and intelligent laboratory systems*, 50(1):1–18, 2000.

[19] Charu C. Aggarwal. *Outlier Analysis*. Springer Publishing Company, Incorporated, 2nd edition, 2016.

[20] Madhu Shashanka, Min-Yi Shen, and Jisheng Wang. User and entity behavior analytics for enterprise security. In *2016 IEEE International Conference on Big Data (Big Data)*, pages 1867–1874. IEEE, 2016.

[21] Markus M Breunig, Hans-Peter Kriegel, Raymond T Ng, and Jörg Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[22] Zengyou He, Xiaofei Xu, and Shengchun Deng. Discovering cluster-based local outliers. *Pattern Recognition Letters*, 24(9-10):1641–1650, 2003.

[23] Mennatallah Amer, Markus Goldstein, and Slim Abdennadher. Enhancing one-class support vector machines for unsupervised anomaly detection. In *Proceedings of the ACM SIGKDD workshop on outlier detection and description*, pages 8–15, 2013.

[24] Bernhard Schölkopf, Robert C Williamson, Alex Smola, John Shawe-Taylor, and John Platt. Support vector method for novelty detection. *Advances in neural information processing systems*, 12:582–588, 1999.

[25] Emmanuel J Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *Journal of the ACM (JACM)*, 58(3):1–37, 2011.

[26] Zhouchen Lin, Minming Chen, and Yi Ma. The augmented lagrange multiplier method for exact recovery of corrupted low-rank matrices. *arXiv preprint arXiv:1009.5055*, 2010.

[27] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, pages 413–422. IEEE, 2008.

[28] Li Sun, Steven Versteeg, Serdar Boztas, and Asha Rao. Detecting anomalous user behavior using an extended isolation forest algorithm: an enterprise case study. *arXiv preprint arXiv:1609.06676*, 2016.

[29] Sahand Hariri, Matias Carrasco Kind, and Robert J Brunner. Extended isolation forest. *arXiv preprint arXiv:1811.02141*, 2018.

[30] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144, 2016.

[31] Scott M Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Advances in neural information processing systems*, pages 4765–4774, 2017.

[32] Mattia Carletti, Matteo Terzi, and Gian Antonio Susto. Interpretable anomaly detection with diffi: Depth-based feature importance for the isolation forest. *arXiv preprint arXiv:2007.11117*, 2020.

[33] Al Perlman. The growing role of machine learning in cybersecurity. `https://www.securityroundtable.org/the-growing-role-of-machine-learning-in-cybersecurity/`, 2019. Accessed: 2020-09.

[34] Soma Halder and Sinan Ozdemir. *Hands-on Machine Learning for Cybersecurity.* Packt Publishing, 2018.

[35] Varonis. What is ueba? `https://www.varonis.com/blog/user-entity-behavior-analytics-ueba/`, 2020. Accessed: 2020-09.

[36] Top 10 ueba security use cases: Compromised user credentials, executive assets monitoring, data exfiltration detection. `https://www.exabeam.com/ueba/top-10-ueba-security-use-cases/`, 2019. Accessed: 2020-09.

[37] Gurucul. User and entity behavior analytics use cases. 2018.

[38] Gurucul. User and entity behavior analytics data sheet. 2020.

[39] Jane Grafton. Abcs of ueba: M is for machine learning. `https://gurucul.com/blog/abcs-of-ueba-m-is-for-machine-learning/`, 2019. Accessed: 2020-09.

[40] Md Salik Parwez, Danda B Rawat, and Moses Garuba. Big data analytics for user-activity analysis and user-anomaly detection in mobile wireless network. *IEEE Transactions on Industrial Informatics*, 13(4):2058–2065, 2017.

[41] Stephanie Condon. Hpe acquires behavioral security analytics firm niara. `https://www.zdnet.com/article/hpe-acquires-behavioral-security-analytics-firm-niara/`, 2017. Accessed: 2020-09.

[42] Xiangyu Xi, Tong Zhang, Wei Ye, Zhao Wen, Shikun Zhang, Dongdong Du, and Qing Gao. An ensemble approach for detecting anomalous user behaviors. *International Journal of Software Engineering and Knowledge Engineering*, 28(11n12):1637–1656, 2018.

[43] Qiaona Hu, Baoming Tang, and Derek Lin. Anomalous user activity detection in enterprise multi-source logs. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 797–803. IEEE, 2017.

[44] Mei-Ling Shyu, Shu-Ching Chen, Kanoksri Sarinnapakorn, and LiWu Chang. A novel anomaly detection scheme based on principal component classifier. Technical report, MIAMI UNIV CORAL GABLES FL DEPT OF ELECTRICAL AND COMPUTER ENGINEERING, 2003.

[45] Scott Lundberg. A game theoretic approach to explain the output of any machine learning model. `https://github.com/slundberg/shap`. Accessed: 2021-03.

This page is intentionally left blank.

Appendices

This page is intentionally left blank.

Appendix A

Risk analysis

This appendix contains the risk analysis conducted for this work.

Due to the exploratory nature of this work, a risk analysis capable of summarizing the most relevant risks was conducted, whose results are presented in table A.1. This analysis occurred without recurring to any specific risk methodology, but rather an informal assessment of the potential issues and their consequences during the planning of the work/internship theme. The table contains the risks prioritized in a decreasing fashion, where each row contains one condition (which is true now) and the respective consequence.

Looking at the conditions identified before the work began, the only one that was actually verified, and that had a significant impact on the results, was the third condition stated, related to the deadlines and dependency on other teams, such as for the infrastructure to be in place. Nevertheless, we were able to include real results and tackle most of the challenges required to reach production.

Table A.1: Risk analysis

Condition	**Consequence**
We use data that is specific of one contact center, and may not representative of the activity of others.	The results obtained may not be reproducible for all Talkdesk's clients.
The rate of false positives regarding the anomalies identified is not neglectable.	Users may be accused of something they did not to, which has a major impact on the confidence in the system.
The release date of the project may fall behind the internship's deadline, due to circumstances external to this work.	We will not be able to include real results in the report.
We did not find any other approach of a successful implementation of UEBA in the contact center domain.	UEBA may be unsuitable for detecting threats in contact centers (the activity of users is hard to profile or baseline)

This page is intentionally left blank.

Appendix B

Plan of activities

This section contains a Gantt diagram of the main activities developed, along the period of time in which the work was conducted.

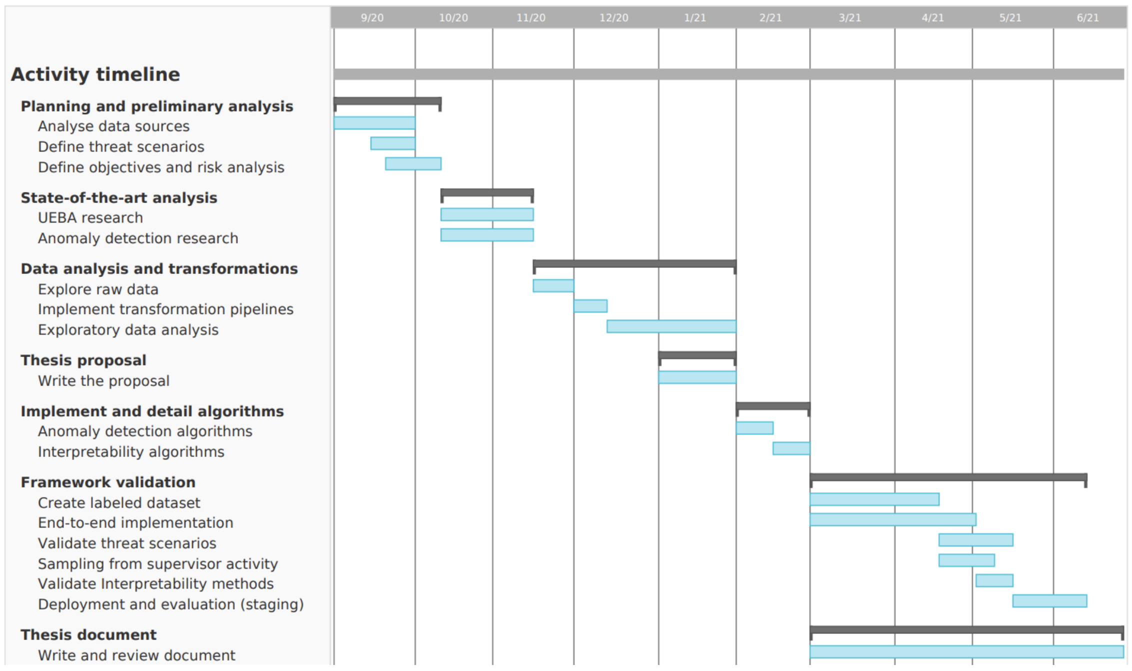


Figure B.1: Gantt diagram containing the timeline of activities conducted.

This page is intentionally left blank.

Appendix C

Results with the labeled dataset

This appendix contains the results of the analysis of the algorithms, using labeled data, as presented in section 5.1. It contains one table per algorithm.

Table C.1: HBOS results

Training set parameters			Model parameters	Metrics						
Baseline period	Feature set	Training contam.	Contam.	F1	ROC	Precision	Recall	FP	FN
30	FS1	max	0.02	0.757	0.831	0.837	0.691	20	46
30	FS2	med	0.02	0.756	0.809	0.959	0.624	4	56
14	FS1	true	0.02	0.741	0.806	0.912	0.624	9	56
14	FS1	max	0.05	0.74	0.829	0.788	0.698	28	45
30	FS1	med	0.02	0.733	0.795	0.947	0.597	5	60
30	FS1	med	0.05	0.733	0.816	0.818	0.664	22	50
30	FS2	med	0.05	0.732	0.805	0.87	0.631	14	55
30	FS1	true	0.02	0.729	0.805	0.862	0.631	15	55
60	FS3	med	0.02	0.728	0.838	0.719	0.738	43	39
30	FS1	max	0.05	0.724	0.826	0.745	0.705	36	44
60	FS2	max	0.02	0.717	0.804	0.819	0.638	21	54
14	FS1	max	0.02	0.714	0.794	0.858	0.611	15	58
60	FS1	max	0.02	0.713	0.807	0.789	0.651	26	52
14	FS1	true	0.05	0.712	0.8	0.817	0.631	21	55
14	FS1	med	0.02	0.711	0.782	0.944	0.57	5	64
60	FS2	true	0.02	0.705	0.792	0.835	0.611	18	58
30	FS1	true	0.05	0.704	0.798	0.797	0.631	24	55
60	FS3	true	0.02	0.704	0.788	0.856	0.597	15	60
30	FS1	med	0.1	0.7	0.833	0.655	0.752	59	37
60	FS1	true	0.02	0.7	0.789	0.833	0.604	18	59
14	FS2	true	0.02	0.7	0.775	0.943	0.557	5	66
60	FS3	med	0.05	0.695	0.855	0.596	0.832	84	25
30	FS2	true	0.02	0.695	0.773	0.922	0.557	7	66
60	FS3	med	0.1	0.695	0.871	0.571	0.886	99	17
14	FS2	true	0.05	0.689	0.772	0.902	0.557	9	66
60	FS2	true	0.1	0.688	0.843	0.604	0.799	78	30
14	FS3	med	0.05	0.681	0.795	0.722	0.644	37	53
60	FS1	true	0.1	0.68	0.834	0.604	0.779	76	33

14	FS1	max	0.1	0.677	0.815	0.641	0.718	60	42
60	FS3	true	0.05	0.677	0.827	0.611	0.758	72	36
30	FS2	med	0.1	0.676	0.801	0.68	0.671	47	49
60	FS2	med	0.02	0.675	0.772	0.825	0.57	18	64
30	FS2	max	0.02	0.672	0.764	0.88	0.544	11	68
60	FS2	max	0.05	0.669	0.801	0.66	0.678	52	48
60	FS1	max	0.05	0.669	0.801	0.66	0.678	52	48
60	FS1	med	0.02	0.667	0.768	0.816	0.564	19	65
14	FS2	max	0.02	0.664	0.754	0.928	0.517	6	72
14	FS2	med	0.02	0.664	0.752	0.95	0.51	4	73
14	FS1	med	0.05	0.662	0.772	0.763	0.584	27	62
30	FS3	max	0.02	0.659	0.777	0.726	0.604	34	59
14	FS1	true	0.1	0.655	0.783	0.681	0.631	44	55
30	FS3	max	0.05	0.655	0.787	0.667	0.644	48	53
30	FS2	true	0.05	0.654	0.764	0.778	0.564	24	65
30	FS1	max	0.1	0.651	0.81	0.586	0.732	77	40
14	FS3	max	0.05	0.65	0.776	0.695	0.611	40	58
60	FS2	med	0.1	0.649	0.809	0.583	0.732	78	40
30	FS1	true	0.1	0.648	0.783	0.66	0.638	49	54
60	FS1	true	0.05	0.648	0.776	0.689	0.611	41	58
14	FS3	max	0.02	0.646	0.76	0.769	0.557	25	66
14	FS3	med	0.02	0.646	0.758	0.781	0.55	23	67
60	FS1	med	0.1	0.645	0.804	0.585	0.718	76	42
14	FS3	med	0.1	0.641	0.788	0.619	0.664	61	50
60	FS2	true	0.05	0.637	0.774	0.657	0.617	48	57
30	FS2	max	0.05	0.637	0.765	0.702	0.584	37	62
60	FS2	max	0.1	0.636	0.804	0.562	0.732	85	40
14	FS2	max	0.05	0.628	0.753	0.732	0.55	30	67
30	FS3	true	0.02	0.628	0.751	0.743	0.544	28	68
30	FS3	true	0.1	0.627	0.778	0.611	0.644	61	53
30	FS3	true	0.05	0.624	0.763	0.662	0.591	45	61
60	FS3	max	0.02	0.624	0.752	0.719	0.55	32	67
60	FS3	max	0.05	0.623	0.766	0.643	0.604	50	59
30	FS3	med	0.05	0.621	0.758	0.672	0.577	42	63
60	FS3	true	0.1	0.613	0.815	0.492	0.812	125	28
60	FS1	med	0.05	0.612	0.753	0.659	0.57	44	64
14	FS3	true	0.1	0.611	0.765	0.605	0.617	60	57
14	FS3	true	0.02	0.611	0.739	0.748	0.517	26	72
60	FS2	med	0.05	0.61	0.754	0.647	0.577	47	63
30	FS3	med	0.1	0.608	0.767	0.588	0.631	66	55
14	FS2	med	0.05	0.608	0.737	0.752	0.51	25	73
14	FS3	true	0.05	0.607	0.747	0.678	0.55	39	67
14	FS2	true	0.1	0.606	0.75	0.656	0.564	44	65
60	FS1	max	0.1	0.605	0.785	0.53	0.705	93	44
14	FS3	max	0.1	0.604	0.771	0.564	0.651	75	52
30	FS3	max	0.1	0.599	0.771	0.551	0.658	80	51
30	FS3	med	0.02	0.59	0.728	0.725	0.497	28	75
14	FS1	med	0.1	0.586	0.748	0.588	0.584	61	62
30	FS2	max	0.1	0.586	0.753	0.57	0.604	68	59
60	FS3	max	0.1	0.585	0.762	0.536	0.644	83	53
30	FS2	true	0.1	0.581	0.741	0.6	0.564	56	65

14	FS2	med	0.1	0.566	0.728	0.608	0.53	51	70
14	FS2	max	0.1	0.564	0.735	0.564	0.564	65	65

Table C.2: Mahalanobis method results

Training set parameters			Model parameters	Metrics						
Baseline period	Feature set	Training contam.	Contam.	F1	ROC	Precision	Recall	FP	FN
14	FS1	true	0.05	0.893	0.915	0.954	0.839	6	24
60	FS1	true	0.05	0.887	0.914	0.94	0.839	8	24
60	FS2	true	0.02	0.876	0.899	0.96	0.805	5	29
14	FS2	true	0.02	0.872	0.896	0.96	0.799	5	30
30	FS1	true	0.02	0.864	0.892	0.952	0.792	6	31
60	FS2	true	0.05	0.863	0.91	0.881	0.846	17	23
14	FS1	true	0.02	0.863	0.889	0.959	0.785	5	32
14	FS2	true	0.05	0.857	0.909	0.869	0.846	19	23
60	FS2	med	0.05	0.848	0.885	0.921	0.785	10	32
30	FS1	true	0.05	0.846	0.902	0.861	0.832	20	25
14	FS1	med	0.05	0.846	0.88	0.935	0.772	8	34
60	FS1	true	0.02	0.842	0.872	0.957	0.752	5	37
60	FS2	med	0.02	0.84	0.867	0.973	0.738	3	39
14	FS1	true	0.1	0.838	0.942	0.747	0.953	48	7
30	FS2	true	0.05	0.833	0.882	0.886	0.785	15	32
30	FS2	med	0.02	0.832	0.863	0.965	0.732	4	40
30	FS2	max	0.02	0.83	0.865	0.948	0.738	6	39
14	FS2	max	0.05	0.83	0.888	0.857	0.805	20	29
14	FS2	med	0.05	0.827	0.876	0.891	0.772	14	34
30	FS1	med	0.02	0.826	0.857	0.973	0.718	3	42
14	FS2	med	0.1	0.824	0.928	0.742	0.926	48	11
14	FS2	max	0.02	0.822	0.854	0.972	0.711	3	43
30	FS2	med	0.05	0.822	0.881	0.855	0.792	20	31
30	FS1	max	0.02	0.821	0.867	0.903	0.752	12	37
30	FS1	med	0.05	0.817	0.875	0.859	0.779	19	33
14	FS1	max	0.02	0.817	0.85	0.972	0.705	3	44
60	FS2	max	0.02	0.817	0.85	0.972	0.705	3	44
14	FS2	med	0.02	0.817	0.85	0.972	0.705	3	44
14	FS1	med	0.1	0.814	0.925	0.726	0.926	52	11
60	FS1	max	0.02	0.812	0.851	0.946	0.711	6	43
14	FS1	med	0.02	0.812	0.847	0.972	0.698	3	45
14	FS1	max	0.05	0.807	0.88	0.815	0.799	27	30
60	FS2	max	0.05	0.804	0.865	0.856	0.758	19	36
14	FS2	true	0.1	0.799	0.909	0.723	0.893	51	16
30	FS2	true	0.1	0.799	0.896	0.751	0.852	42	22
60	FS1	med	0.05	0.797	0.848	0.906	0.711	11	43
14	FS2	max	0.1	0.791	0.918	0.69	0.926	62	11
30	FS2	max	0.05	0.79	0.882	0.762	0.819	38	27
14	FS3	med	0.05	0.779	0.91	0.68	0.913	64	13
14	FS1	max	0.1	0.779	0.91	0.68	0.913	64	13
60	FS1	true	0.1	0.778	0.903	0.689	0.893	60	16
60	FS2	true	0.1	0.776	0.907	0.678	0.906	64	14
60	FS3	true	0.05	0.767	0.897	0.677	0.886	63	17
60	FS3	true	0.1	0.766	0.912	0.65	0.933	75	10
14	FS3	med	0.1	0.763	0.92	0.633	0.96	83	6

60	FS2	med	0.1	0.762	0.887	0.684	0.859	59	21
30	FS3	true	0.05	0.761	0.895	0.667	0.886	66	17
14	FS3	med	0.02	0.76	0.89	0.674	0.872	63	19
14	FS3	max	0.05	0.758	0.896	0.658	0.893	69	16
14	FS3	true	0.05	0.753	0.901	0.642	0.913	76	13
30	FS1	true	0.1	0.752	0.886	0.665	0.866	65	20
60	FS1	max	0.05	0.75	0.853	0.735	0.765	41	35
60	FS1	med	0.1	0.75	0.872	0.687	0.826	56	26
30	FS1	max	0.05	0.749	0.861	0.711	0.792	48	31
30	FS3	med	0.05	0.749	0.887	0.657	0.872	68	19
30	FS3	true	0.1	0.743	0.899	0.623	0.919	83	12
60	FS3	max	0.05	0.742	0.887	0.642	0.879	73	18
14	FS3	max	0.1	0.741	0.912	0.603	0.96	94	6
30	FS2	true	0.02	0.741	0.799	0.957	0.604	4	59
30	FS1	med	0.1	0.741	0.876	0.66	0.846	65	23
30	FS2	med	0.1	0.741	0.876	0.66	0.846	65	23
30	FS3	true	0.02	0.739	0.867	0.674	0.819	59	27
14	FS3	true	0.02	0.736	0.861	0.678	0.805	57	29
30	FS3	max	0.02	0.735	0.871	0.654	0.839	66	24
60	FS2	max	0.1	0.733	0.879	0.635	0.866	74	20
60	FS3	max	0.02	0.732	0.866	0.658	0.826	64	26
60	FS3	med	0.1	0.732	0.889	0.618	0.899	83	15
60	FS3	med	0.05	0.732	0.868	0.653	0.832	66	25
60	FS3	true	0.02	0.731	0.862	0.665	0.812	61	28
30	FS3	med	0.1	0.729	0.901	0.596	0.94	95	9
30	FS2	max	0.1	0.725	0.885	0.61	0.893	85	16
30	FS3	med	0.02	0.725	0.86	0.654	0.812	64	28
14	FS3	max	0.02	0.723	0.857	0.656	0.805	63	29
30	FS1	max	0.1	0.723	0.884	0.607	0.893	86	16
60	FS1	max	0.1	0.722	0.871	0.626	0.852	76	22
60	FS1	med	0.02	0.72	0.784	0.977	0.57	2	64
30	FS3	max	0.05	0.714	0.878	0.597	0.886	89	17
30	FS3	max	0.1	0.714	0.897	0.573	0.946	105	8
60	FS3	max	0.1	0.714	0.895	0.576	0.94	103	9
60	FS3	med	0.02	0.696	0.826	0.665	0.732	55	40
14	FS3	true	0.1	0.678	0.872	0.54	0.913	116	13

Table C.3: Isolation forest results

Training set parameters			Model parameters	Metrics						
Baseline period	Feature set	Training contam.	Contam.	F1	ROC	Precision	Recall	FP	FN
14	FS2	true	0.1	0.861	0.927	0.831	0.893	27	16
14	FS1	true	0.1	0.851	0.915	0.838	0.866	25	20
14	FS2	true	0.05	0.849	0.881	0.943	0.772	7	34
30	FS1	true	0.05	0.847	0.89	0.902	0.799	13	30
30	FS2	true	0.1	0.84	0.931	0.774	0.919	40	12
30	FS1	true	0.1	0.834	0.919	0.782	0.893	37	16
60	FS2	true	0.05	0.829	0.874	0.905	0.765	12	35
60	FS1	true	0.05	0.829	0.878	0.885	0.779	15	33
30	FS2	true	0.05	0.827	0.869	0.918	0.752	10	37
60	FS1	true	0.1	0.825	0.915	0.772	0.886	39	17
14	FS3	true	0.05	0.822	0.886	0.839	0.805	23	29
14	FS2	med	0.1	0.822	0.907	0.782	0.866	36	20
60	FS2	med	0.1	0.822	0.914	0.767	0.886	40	17
60	FS2	true	0.1	0.821	0.92	0.75	0.906	45	14
14	FS1	true	0.05	0.818	0.857	0.939	0.725	7	41
30	FS3	med	0.05	0.813	0.888	0.808	0.819	29	27
14	FS1	med	0.1	0.813	0.893	0.795	0.832	32	25
60	FS2	med	0.05	0.812	0.856	0.923	0.725	9	41
14	FS1	max	0.05	0.809	0.864	0.875	0.752	16	37
14	FS2	max	0.05	0.801	0.856	0.886	0.732	14	40
60	FS3	med	0.05	0.799	0.896	0.751	0.852	42	22
14	FS1	med	0.05	0.797	0.843	0.929	0.698	8	45
14	FS2	med	0.05	0.792	0.835	0.953	0.678	5	48
30	FS2	med	0.05	0.789	0.839	0.92	0.691	9	46
14	FS2	max	0.1	0.784	0.887	0.735	0.839	45	24
30	FS2	med	0.1	0.778	0.892	0.711	0.859	52	21
60	FS3	true	0.05	0.778	0.896	0.703	0.872	55	19
14	FS1	max	0.1	0.775	0.877	0.735	0.819	44	27
30	FS3	max	0.05	0.774	0.873	0.745	0.805	41	29
30	FS1	med	0.1	0.766	0.873	0.725	0.812	46	28
14	FS3	med	0.1	0.766	0.916	0.644	0.946	78	8
14	FS2	true	0.02	0.763	0.815	0.95	0.638	5	54
30	FS1	med	0.05	0.762	0.823	0.892	0.664	12	50
30	FS3	true	0.05	0.759	0.892	0.668	0.879	65	18
60	FS2	max	0.05	0.758	0.842	0.794	0.725	28	41
60	FS1	med	0.05	0.757	0.827	0.856	0.678	17	48
60	FS2	max	0.1	0.756	0.895	0.655	0.893	70	16
14	FS3	med	0.05	0.752	0.868	0.699	0.812	52	28
14	FS3	true	0.1	0.752	0.905	0.633	0.926	80	11
60	FS3	true	0.1	0.752	0.911	0.624	0.946	85	8
60	FS1	med	0.1	0.751	0.857	0.725	0.779	44	33
60	FS3	max	0.05	0.749	0.88	0.668	0.852	63	22
14	FS1	max	0.02	0.746	0.809	0.913	0.631	9	55
60	FS2	max	0.02	0.741	0.806	0.912	0.624	9	56
30	FS2	max	0.05	0.74	0.835	0.764	0.718	33	42

14	FS3	max	0.1	0.74	0.907	0.608	0.946	91	8
60	FS3	med	0.1	0.738	0.908	0.602	0.953	94	7
30	FS3	true	0.1	0.738	0.906	0.605	0.946	92	8
30	FS3	med	0.1	0.734	0.905	0.6	0.946	94	8
30	FS3	true	0.02	0.732	0.812	0.836	0.651	19	52
14	FS3	true	0.02	0.73	0.794	0.937	0.597	6	60
60	FS3	max	0.1	0.722	0.896	0.589	0.933	97	10
60	FS3	max	0.02	0.719	0.814	0.775	0.671	29	49
30	FS1	true	0.02	0.718	0.781	0.988	0.564	1	65
60	FS1	true	0.02	0.718	0.781	0.988	0.564	1	65
14	FS3	max	0.05	0.717	0.863	0.629	0.832	73	25
14	FS3	max	0.02	0.716	0.807	0.795	0.651	25	52
14	FS1	true	0.02	0.716	0.779	1.0	0.557	0	66
30	FS2	max	0.1	0.715	0.862	0.626	0.832	74	25
14	FS3	med	0.02	0.709	0.799	0.81	0.631	22	55
30	FS2	true	0.02	0.707	0.774	0.988	0.55	1	67
30	FS3	max	0.02	0.704	0.8	0.785	0.638	26	54
60	FS1	max	0.1	0.7	0.835	0.649	0.758	61	36
60	FS1	max	0.05	0.7	0.805	0.748	0.658	33	51
60	FS3	true	0.02	0.695	0.787	0.818	0.604	20	59
30	FS3	max	0.1	0.686	0.869	0.556	0.893	106	16
60	FS2	true	0.02	0.678	0.758	0.987	0.517	1	72
60	FS2	med	0.02	0.673	0.754	0.987	0.51	1	73
60	FS3	med	0.02	0.672	0.771	0.817	0.57	19	64
30	FS1	max	0.05	0.671	0.789	0.718	0.631	37	55
30	FS1	max	0.1	0.667	0.814	0.617	0.725	67	41
30	FS3	med	0.02	0.664	0.767	0.808	0.564	20	65
14	FS2	max	0.02	0.661	0.751	0.938	0.51	5	73
30	FS1	max	0.02	0.656	0.756	0.859	0.53	13	70
30	FS2	max	0.02	0.655	0.752	0.895	0.517	9	72
14	FS2	med	0.02	0.652	0.744	0.973	0.49	2	76
30	FS2	med	0.02	0.646	0.742	0.948	0.49	4	76
60	FS1	max	0.02	0.638	0.744	0.872	0.503	11	74
30	FS1	med	0.02	0.628	0.732	0.946	0.47	4	79
14	FS1	med	0.02	0.627	0.73	0.972	0.463	2	80
60	FS1	med	0.02	0.609	0.722	0.944	0.45	4	82

Table C.4: PCA results

Training set parameters			Model parameters	Metrics						
Baseline period	Feature set	Training contam.	Contam.	F1	ROC	Precision	Recall	FP	FN
14	FS1	med	0.05	0.932	0.951	0.951	0.913	7	13
60	FS2	true	0.05	0.931	0.964	0.916	0.946	13	8
14	FS2	max	0.05	0.93	0.958	0.927	0.933	11	10
14	FS1	true	0.05	0.93	0.958	0.927	0.933	11	10
14	FS2	med	0.05	0.928	0.951	0.944	0.913	8	13
30	FS2	true	0.05	0.928	0.966	0.904	0.953	15	7
60	FS2	med	0.05	0.927	0.958	0.921	0.933	12	10
14	FS2	true	0.05	0.926	0.952	0.932	0.919	10	12
14	FS2	true	0.1	0.92	0.966	0.883	0.96	19	6
14	FS1	med	0.1	0.918	0.958	0.897	0.94	16	9
30	FS1	true	0.05	0.918	0.958	0.897	0.94	16	9
30	FS2	med	0.05	0.914	0.952	0.902	0.926	15	11
60	FS1	true	0.05	0.914	0.955	0.897	0.933	16	10
14	FS1	max	0.05	0.914	0.955	0.897	0.933	16	10
14	FS2	max	0.02	0.913	0.935	0.949	0.879	7	18
60	FS1	med	0.05	0.913	0.95	0.907	0.919	14	12
14	FS2	med	0.1	0.912	0.957	0.886	0.94	18	9
14	FS1	true	0.1	0.908	0.963	0.861	0.96	23	6
30	FS1	med	0.05	0.907	0.948	0.895	0.919	16	12
60	FS2	true	0.02	0.905	0.925	0.955	0.859	6	21
30	FS2	true	0.02	0.901	0.922	0.955	0.852	6	22
30	FS1	true	0.02	0.9	0.919	0.962	0.846	5	23
60	FS1	true	0.02	0.9	0.919	0.962	0.846	5	23
30	FS2	max	0.02	0.899	0.936	0.905	0.893	14	16
14	FS1	true	0.02	0.899	0.917	0.969	0.839	4	24
14	FS2	true	0.02	0.897	0.918	0.955	0.846	6	23
60	FS2	max	0.05	0.897	0.95	0.863	0.933	22	10
30	FS2	true	0.1	0.897	0.963	0.837	0.966	28	5
60	FS2	true	0.1	0.897	0.963	0.837	0.966	28	5
60	FS2	max	0.02	0.896	0.926	0.928	0.866	10	20
14	FS1	max	0.02	0.895	0.913	0.969	0.832	4	25
60	FS1	max	0.02	0.89	0.917	0.94	0.846	8	23
14	FS2	max	0.1	0.889	0.951	0.843	0.94	26	9
60	FS1	true	0.1	0.885	0.957	0.822	0.96	31	6
60	FS2	med	0.1	0.883	0.949	0.833	0.94	28	9
30	FS2	med	0.1	0.881	0.949	0.828	0.94	29	9
30	FS2	max	0.05	0.88	0.946	0.832	0.933	28	10
14	FS1	max	0.1	0.879	0.951	0.82	0.946	31	8
30	FS1	true	0.1	0.877	0.955	0.808	0.96	34	6
60	FS1	max	0.05	0.876	0.943	0.831	0.926	28	11
60	FS2	med	0.02	0.875	0.896	0.967	0.799	4	30
60	FS1	med	0.1	0.873	0.949	0.81	0.946	33	8
30	FS2	med	0.02	0.873	0.898	0.952	0.805	6	29
30	FS1	max	0.02	0.87	0.915	0.888	0.852	16	22
14	FS2	med	0.02	0.868	0.892	0.959	0.792	5	31

30	FS1	med	0.02	0.867	0.89	0.967	0.785	4	32
60	FS1	med	0.02	0.866	0.887	0.975	0.779	3	33
14	FS1	med	0.02	0.865	0.884	0.983	0.772	2	34
30	FS1	med	0.1	0.862	0.946	0.792	0.946	37	8
30	FS1	max	0.05	0.854	0.937	0.793	0.926	36	11
60	FS2	max	0.1	0.843	0.939	0.765	0.94	43	9
60	FS1	max	0.1	0.829	0.937	0.738	0.946	50	8
30	FS1	max	0.1	0.822	0.935	0.727	0.946	53	8
30	FS2	max	0.1	0.819	0.931	0.725	0.94	53	9
14	FS3	med	0.02	0.807	0.932	0.7	0.953	61	7
14	FS3	true	0.05	0.805	0.931	0.696	0.953	62	7
14	FS3	med	0.05	0.804	0.936	0.689	0.966	65	5
30	FS3	true	0.02	0.8	0.925	0.697	0.94	61	9
60	FS3	true	0.02	0.797	0.922	0.695	0.933	61	10
14	FS3	true	0.02	0.795	0.919	0.697	0.926	60	11
14	FS3	med	0.1	0.793	0.937	0.667	0.98	73	3
60	FS3	true	0.05	0.791	0.932	0.67	0.966	71	5
30	FS3	true	0.05	0.788	0.928	0.668	0.96	71	6
14	FS3	true	0.1	0.788	0.933	0.662	0.973	74	4
14	FS3	max	0.05	0.786	0.932	0.659	0.973	75	4
60	FS3	med	0.05	0.784	0.927	0.662	0.96	73	6
30	FS3	med	0.05	0.784	0.927	0.662	0.96	73	6
14	FS3	max	0.02	0.783	0.913	0.682	0.919	64	12
30	FS3	med	0.02	0.781	0.906	0.691	0.899	60	15
30	FS3	true	0.1	0.781	0.933	0.649	0.98	79	3
30	FS3	max	0.02	0.78	0.923	0.66	0.953	73	7
60	FS3	max	0.05	0.78	0.93	0.65	0.973	78	4
60	FS3	med	0.02	0.779	0.905	0.687	0.899	61	15
60	FS3	true	0.1	0.777	0.931	0.643	0.98	81	3
60	FS3	max	0.02	0.776	0.917	0.66	0.94	72	9
14	FS3	max	0.1	0.773	0.928	0.642	0.973	81	4
30	FS3	max	0.05	0.773	0.928	0.642	0.973	81	4
60	FS3	med	0.1	0.771	0.927	0.639	0.973	82	4
30	FS3	med	0.1	0.769	0.926	0.636	0.973	83	4
60	FS3	max	0.1	0.74	0.915	0.597	0.973	98	4
30	FS3	max	0.1	0.734	0.913	0.589	0.973	101	4

Table C.5: Autoencoder results

Training set parameters			Model parameters	Metrics						
Baseline period	Feature set	Training contam.	Contam.	F1	ROC	Precision	Recall	FP	FN
30	FS2	true	0.05	0.947	0.973	0.935	0.96	10	6
14	FS1	true	0.05	0.944	0.969	0.934	0.953	10	7
60	FS2	true	0.05	0.944	0.972	0.929	0.96	11	6
14	FS1	med	0.05	0.936	0.963	0.933	0.94	10	9
14	FS2	true	0.05	0.936	0.963	0.933	0.94	10	9
30	FS2	med	0.05	0.936	0.96	0.939	0.933	9	10
30	FS1	true	0.05	0.935	0.97	0.911	0.96	14	6
60	FS2	med	0.05	0.933	0.959	0.933	0.933	10	10
14	FS2	med	0.05	0.932	0.951	0.951	0.913	7	13
60	FS1	true	0.05	0.932	0.969	0.905	0.96	15	6
30	FS1	med	0.05	0.927	0.96	0.915	0.94	13	9
60	FS1	med	0.05	0.927	0.96	0.915	0.94	13	9
14	FS1	max	0.02	0.927	0.945	0.957	0.899	6	15
60	FS2	max	0.02	0.922	0.946	0.938	0.906	9	14
30	FS2	max	0.02	0.921	0.956	0.908	0.933	14	10
60	FS1	max	0.02	0.914	0.94	0.937	0.893	9	16
30	FS2	true	0.02	0.913	0.935	0.949	0.879	7	18
14	FS2	max	0.02	0.912	0.932	0.956	0.872	6	19
30	FS1	max	0.02	0.911	0.954	0.891	0.933	17	10
60	FS2	max	0.05	0.908	0.953	0.885	0.933	18	10
14	FS2	true	0.02	0.905	0.925	0.955	0.859	6	21
60	FS2	true	0.02	0.905	0.925	0.955	0.859	6	21
30	FS2	max	0.05	0.901	0.956	0.86	0.946	23	8
30	FS1	true	0.02	0.901	0.922	0.955	0.852	6	22
60	FS1	true	0.02	0.9	0.919	0.962	0.846	5	23
14	FS1	max	0.05	0.898	0.956	0.855	0.946	24	8
14	FS1	true	0.02	0.897	0.918	0.955	0.846	6	23
14	FS1	med	0.02	0.891	0.91	0.969	0.826	4	26
14	FS2	med	0.02	0.888	0.909	0.961	0.826	5	26
60	FS2	med	0.02	0.888	0.909	0.961	0.826	5	26
30	FS1	max	0.05	0.883	0.949	0.833	0.94	28	9
14	FS2	max	0.05	0.882	0.954	0.821	0.953	31	7
60	FS1	max	0.05	0.881	0.949	0.828	0.94	29	9
30	FS2	med	0.02	0.877	0.902	0.953	0.812	6	28
30	FS2	true	0.1	0.875	0.954	0.803	0.96	35	6
30	FS1	med	0.02	0.872	0.896	0.96	0.799	5	30
14	FS2	true	0.1	0.87	0.948	0.806	0.946	34	8
14	FS2	med	0.1	0.87	0.948	0.806	0.946	34	8
60	FS2	true	0.1	0.869	0.953	0.794	0.96	37	6
60	FS1	med	0.02	0.867	0.89	0.967	0.785	4	32
60	FS2	med	0.1	0.867	0.952	0.79	0.96	38	6
30	FS2	med	0.1	0.867	0.952	0.79	0.96	38	6
14	FS2	max	0.1	0.861	0.951	0.781	0.96	40	6
14	FS1	med	0.1	0.859	0.95	0.777	0.96	41	6
30	FS1	true	0.1	0.856	0.949	0.773	0.96	42	6

60	FS1	true	0.1	0.856	0.949	0.773	0.96	42	6
14	FS1	true	0.1	0.849	0.947	0.761	0.96	45	6
60	FS1	med	0.1	0.849	0.947	0.761	0.96	45	6
30	FS1	med	0.1	0.841	0.945	0.749	0.96	48	6
60	FS2	max	0.1	0.839	0.944	0.745	0.96	49	6
14	FS1	max	0.1	0.839	0.944	0.745	0.96	49	6
30	FS2	max	0.1	0.824	0.94	0.722	0.96	55	6
60	FS1	max	0.1	0.817	0.938	0.711	0.96	58	6
14	FS3	med	0.02	0.805	0.931	0.696	0.953	62	7
14	FS3	true	0.05	0.801	0.933	0.688	0.96	65	6
30	FS3	true	0.02	0.8	0.925	0.697	0.94	61	9
14	FS3	med	0.05	0.799	0.936	0.678	0.973	69	4
30	FS1	max	0.1	0.799	0.932	0.684	0.96	66	6
60	FS3	true	0.02	0.797	0.922	0.695	0.933	61	10
30	FS3	true	0.05	0.793	0.932	0.673	0.966	70	5
60	FS3	true	0.05	0.793	0.932	0.673	0.966	70	5
14	FS3	true	0.02	0.793	0.919	0.693	0.926	61	11
60	FS3	max	0.02	0.791	0.932	0.67	0.966	71	5
14	FS3	max	0.05	0.79	0.934	0.665	0.973	73	4
30	FS3	max	0.02	0.789	0.931	0.667	0.966	72	5
30	FS3	med	0.05	0.789	0.931	0.667	0.966	72	5
60	FS3	med	0.05	0.788	0.928	0.668	0.96	71	6
60	FS3	max	0.05	0.788	0.933	0.662	0.973	74	4
14	FS3	max	0.02	0.787	0.921	0.676	0.94	67	9
30	FS3	med	0.02	0.786	0.912	0.69	0.913	61	13
14	FS3	med	0.1	0.785	0.934	0.655	0.98	77	3
14	FS3	true	0.1	0.782	0.931	0.653	0.973	77	4
30	FS3	max	0.05	0.78	0.93	0.65	0.973	78	4
60	FS3	true	0.1	0.779	0.932	0.646	0.98	80	3
60	FS3	med	0.1	0.777	0.929	0.647	0.973	79	4
30	FS3	med	0.1	0.775	0.928	0.644	0.973	80	4
30	FS3	true	0.1	0.775	0.93	0.64	0.98	82	3
60	FS3	med	0.02	0.775	0.904	0.68	0.899	63	15
14	FS3	max	0.1	0.769	0.926	0.636	0.973	83	4
60	FS3	max	0.1	0.755	0.921	0.617	0.973	90	4
30	FS3	max	0.1	0.746	0.918	0.604	0.973	95	4