The Journal of Supercomputing (2023) 79:11660–11683
https://doi.org/10.1007/s11227-023-05049-x

An ensemble-based framework for user behaviour anomaly detection and classification for cybersecurity

Gianluigi Folino[1] · Carla Otranto Godano[2] · Francesco Sergio Pisani[1]

Accepted: 5 January 2023 / Published online: 17 January 2023

Abstract

Nowadays, the speed of the user and application logs is so quick that it is almost impossible to analyse them in real time without using high-performance systems and platforms. In cybersecurity, human behaviour is responsible directly or indirectly for the most common attacks (i.e. ransomware and phishing). To monitor user behaviour, it is necessary to process fast user logs coming from different and heterogeneous sources, having part of the data or some entire sources missing. A framework based on the elastic stack (ELK) to process and store log data in real time from different users and applications is proposed for this aim. This system generates an ensemble of models to classify user behaviour and detect anomalies in real time, exploiting the advantages of the ELK-based software architecture and of the Kubernetes platform. In addition, a distributed evolutionary algorithm is used to classify the users by exploiting their digital footprints derived from many data sources. Experiments conducted on two real-life data sets verify the approach's goodness in detecting anomalies in user behaviour, coping with missing data and lowering the number of false alarms.

Keywords Cybersecurity · Ensemble learning · Anomaly detection · Behavior analytics

✉ Gianluigi Folino
gianluigi.folino@icar.cnr.it

Carla Otranto Godano
carla.otrantogodano@hfactorsec.com

Francesco Sergio Pisani
francescosergio.pisani@icar.cnr.it

1 Department ICAR-CNR, via Bucci 7-9C, 87036 Rende, CS, Italy

2 HFactor Security, 87036 Rende, CS, Italy

1 Introduction

In the last few years, the consideration of many industries and governments towards cyber security risks has been increasing [1], as cybercrime seriously threatens national governments and the economy of many industries. Therefore, suitable and timely countermeasures must be undertaken to protect security vulnerabilities and weaknesses of the systems from potential attacks, to minimise all the risks. In addition, computer network activities, human actions, etc., generate large amounts of data, which must be considered when designing systems and frameworks for cybersecurity protection.

User behaviour could enable many kinds of vulnerability; for example, they can use easy-to-guess passwords for multiple work and personal applications and have excessive trust in social networks and usage of technologies [2, 3]. In 2021, by the IBM Threat Intelligence Index [1], ransomware was the top attack type (21%), and the percentage of attacks exploiting phishing for initial access was about 41%, both caused mainly by user behaviour.

Often, these behaviours or the consequent vulnerabilities are analysed when the attack has already happened; on the contrary, it is necessary to implement a proactive approach to avoid these vulnerabilities being enabled. Therefore, systems for operating with security weaknesses derived from the human factor must consider several critical aspects, such as profiling users for better and more focused actions, analysing large logs in real time and working efficiently in the case of missing data.

Distributed data mining and machine learning techniques could be used to fight efficiently and alleviate the effect or prevent cybercriminals' actions, especially in the presence of large data sets [4]. In particular, classification is used efficiently for many cybersecurity applications, i.e. classification of user behaviour, risk and attack analysis, intrusion detection systems, etc. However, in this particular domain, different data sets often have a different number of features, and each attribute could have different importance and cost. Furthermore, the entire system must also work if some features are missing. Therefore, a single classification algorithm performing well for all the data sets would be unlikely, especially in the presence of changes and with constraints of real time and scalability [5]. In the ensemble learning paradigm [6, 7], multiple classification models are trained by a predictive algorithm, and then their predictions are combined to classify new tuples. This paradigm presents several advantages with regard to using a single model, i.e. it reduces the variance of the error, the bias and the dependence on a single data set and works well in the case of unbalanced classes; furthermore, the ensemble can be built incrementally and can be easily implemented on a distributed environment.

In general, the classification or the clustering of user profiles is often used as a preliminary task to improve the detection of anomalous user behaviours and detect possible anomalies. Indeed, in supervised (or semi-supervised) anomaly detection methods, the classification task is used to separate the normal from anomalous behaviour of the users. The main difference between the two techniques is that the

[1] https://www.ibm.com/security/data-breach/threat-intelligence/.

algorithm must analyse a stream of data containing normal and anomalous behaviours in supervised anomaly detection. In contrast, in the semi-supervised technique, the stream contains only data concerning normal behaviour. Our system follows the semi-supervised approach, as, in the real world, it would be tough to have a sufficient number of "real" anomalies to train the classification algorithm.

However, because anomaly detection or classification tasks are effective, efficient data indexing must be employed to process heterogeneous and often unbalanced data logs in real time. In addition, these algorithm needs to search and query the big data correlated to the user behaviour with stringent requirements for actual full-text searches over massive data sets.

To overcome the above-cited problematics, we propose a framework based on the elastic stack to process and store the data from the different users and generate an ensemble of classifiers to classify user behaviour and exploit this classification to detect anomalies in their behaviour efficiently. In practice, the system uses the high-performance architecture provided by ELK, running on top of a Kubernetes-based platform and adopts a distributed evolutionary algorithm for classifying the user based on their digital footprints, derived from many logs. In addition, as a new resulting task, the framework permits the individuation of the anomalies in user behaviour. Indeed, the classification algorithm previously introduced in [8] is here used as a preliminary step for identifying likely anomalies by associating a class of risk to all the tuples differing by a predefined threshold by the usual behaviour of the user/group. Then, the anomaly detection task is reformulated as a combination of user/group identification tasks, following the principle that the lower the probability that a digital footprint belongs to its corresponding user/group, the more anomalous behaviour. Experiments conducted on two real-life data sets verify the approach's goodness in detecting anomalies in user behaviour, coping with missing data and lowering the number of false alarms.

The contribution of our system to the analysis of user behaviour for cybersecurity is different from the other works available in the literature, shown in Sect. 2, and can be summarised as follows:

(a) it adopts the ELK stack and Kubernetes to guarantee real-time processing of the user logs;
(b) it can efficiently handle missing and unbalanced sources of data;
(c) it can integrate digital footprints coming from many sources of data in an incremental way;
(d) it can be used for different tasks in preventing cybersecurity problems correlated to the human factor, such as the classification of user profiles and risks and anomaly detection of misuse/abuse user behaviour.

The rest of the paper is organised as follows: Sect. 3 introduces some background information concerning the problem of missing data for classifying user profiles and the approach based on an ensemble of classifiers for classifying the users; Sect. 4 describes the issues correlated to the problem of anomaly detection from the different sources of data (digital footprints) representing the behaviour of the users; in Sect. 5, the ELK stack is introduced together with the main feature of our software architecture; the meta-ensemble approach used to classify the users, and the anomaly detection algorithm is illustrated in Sect. 6; Sect. 7 describes the experiments conducted on two real-life data sets to verify the goodness of the approach in detecting anomalies in the user behaviour, coping with missing data and reducing the false alarms. Finally, Sect. 8 concludes the work.

2 Related works

Recently, in the literature, there has been a growing interest in the task of monitoring user behaviour and actions and using machine learning-based approaches to analyse the resulting logs to minimise or prevent cybersecurity risks or frauds [9, 10].

Most of the works originate from how the user interacts with the computer, the web or social networks. For instance, the authors of [11] exploit information such as the mouse speed, distance, angles and the number of clicks during a user session for user identification and masquerade detection. Adopting the SVM (support vector machine) machine learning algorithm permits a detection rate of up to 96% and a few false positives. This approach has the merit of first introducing the analysis of the data coming from the interaction with a GUI. Still, it does not cope with missing features and different data sources.

Tabia et al. [12] profiled user (normal) behaviour not only by considering computer usage but also network resources. They improve the generalisation performance of a decision tree classification model by using a modified algorithm based on the minimum description length (MDL) principle. As in the previous work, missing data are not considered, and neither is the approach apt to work with fast streams of logs.

In [13], the authors cope with the task of anomaly detection based on the normal usage patterns of profiled users during the different sessions. Mainly, the user behaviour is modelled by monitoring application usage, application performance (CPU and memory), the websites a user visited, the number of windows a user opened and their typing habits. Experimental results demonstrate that physical-related features are relevant to profiling user behaviour and that combining these characteristics can significantly decrease the detection time. This approach improves the efficiency in handling data logs but, differently from our system, does not consider different data sources and does not take advantage of the information on groups of users.

The work by Corney et al. [14] is based on monitoring computer system logs (application usage). A prototype software to build user profiles and identify anomalous events was developed in which the user profiles model the use of programs in a computer system. Furthermore, a promising technique to reduce the number of false positives by grouping the related applications was also introduced. The final results suggest that the number of generated false alerts is significantly reduced when a growing time window is used for user profile training and when the abstraction into the groups of applications is used. The efficiency of this approach is improved by the usage of data collecting and the reduction, and it has a good idea of considering different types of windows of observation (constant, growing and moving) and introduces the idea of considering groups of users, but it is only left for future works.

Harilal et al. [15] built a data set of 24 users from many heterogeneous data sources (i.e. mouse, keyboard, processes and file-system access) with a mix of regular and malicious activities for testing algorithms for masquerader and traitor activities. They conducted several statistics to analyse this data set, but no advanced machine learning technique was adopted.

The authors of [16] introduce the usage of three types of data sets based on user log data: the user's daily activity summary, e-mail contents topic distribution and the user's weekly e-mail communication history. Then, anomaly detection models are independently trained on each data set. Experimental results indicate that the proposed framework can work well for imbalanced data sets with only a few insider threats, and no domain experts' knowledge is provided. The limit of this approach is that the developed models are not combined with an ensemble or something else as our method; therefore, they can hardly handle missing data sources.

To summarise, the above-described papers have the merit of introducing some promising data sources for monitoring user behaviour. Still, they do not cope with real-world problems, such as scalability issues, data streams and missing sources/features.

The work in [17] developed a simple anomaly detection system based on a statistic test executed on a stream of Linux commands, with particular attention to unpopular commands to discern the behaviour of masquerade users. The paper is one of the first to introduce the idea of analysing a stream of data to monitor users; however, the methodology is very basic and not apt to handle complex and real-time streams of data.

Iglesias et al. [18] developed the EvABCD (EVolving Agent behaviour Classification based on Distributions of relevant events) system, which models the user behaviour using the sequence of Unix commands they types in a session, transformed into a distribution of appropriate subsequences of commands to find a profile that defines its behaviour. They considered four classes of users (based on their ability in computer programming), and the prototypes evolved taking into account the new information collected online from a data stream. However, the problem of missing data and processing high-speed data streams are not considered.

Prarthana et al. [19] incorporate contextual and behavioural aspects of the data for user anomaly detection, from a stream of events and network traffic, with multiple dimensions. To overcome the complexity of sparse and unbalanced data, their system incorporates On-Line Analytical Processing (OLAP)-based data analysis with multiscale visualisation for exploratory discovery and a multidimensional statistical test. Experiments performed on real data demonstrate that detection efficiency improves with the increased dimensionality of the tests, which also increases the computational complexity. Therefore, more efficient methods should be employed to operate with real-time data streams.

The authors of [20] try to cope with high-class imbalance, changing target concepts and other real-time issues (heterogeneous attributes, cold start problem, etc.) of anomaly detection by using a user-centred algorithm operating with identity and access management (IAM) real logs. Experimental results on the CERT insider data set show that two different methods must be used for masquerades and discovering malicious users. In the former case, user identification is sufficient, while for the latter, it is necessary to introduce graph features representing user context and relations to other entities. However, even this approach is hardly helpful in the presence of missing features.

To summarise, the papers mentioned above try to address some issues of real-world processing of monitoring user behaviour; however, none is adequate for providing a high-performance system able to process data streams in the presence of unbalanced and missing data.

3 Missing data and ensemble of classifiers

This section introduces the issues and the main methods to cope with missing data and the ensemble-based classifier used in our approach.

For our task, it is essential to profile users when working on their computer or navigating on the Web, or to use social networks; unfortunately, the sources of data, often defined as digital footprints, are heterogeneous and often missing; typical examples are: command line calls issued by users, system calls, database/file accesses, application and web usage logs. Therefore, some techniques for processing and coping with missing data must be introduced. Some main patterns are commonly used for treating missing features: missing completely at random (MCAR), to describe data in which the complete cases are a random sample of the original data set, i.e. the probability of a feature being missing is independent of the value of that or any other feature in the data set; missing at random (MAR) describe data that are missing for reasons related to ultimately observed variables in the data set. Finally, the MNAR (not missing at random) case considers the probability that an entry will be missing depending on both observed and unobserved values in the data set. Therefore, even if MCAR is easier to handle, we try to cope with the MAR case, as it is a more realistic model suitable for many real-world applications, i.e. the cybersecurity scenario.

Data mining and, in particular, classification algorithms must handle the problem of missing values on valuable features. The presence of missing features complicates the classification process, as the effective prediction may depend heavily on how missing values are treated. Typically, missing values are present in the training and testing data, i.e. the same data sources are not provided for all the users. However, in our approach, without any loss of generality, we suppose that the training data set is complete. Even in the case of the presence of a moderate number of tuples presenting missing data, it can be reported to the previous point simply by deleting all the incomplete tuples. However, handling missing data by eliminating examples with missing data will bias results if the remaining cases are not representative of the entire sample. Therefore, different techniques can be used to handle these missing features (see [21] for a detailed list of them). In addition to the strategy mentioned above to remove any tuple with missing values, we remember the option of using a classification algorithm, which can deal with missing values in the training phase and the strategy of imputing all missing values before training the classification algorithm, i.e. replace the missing value with a meaningful estimate.

Indeed, we are more interested in handling groups of missing features in our particular problem. Consequently, we focus on building an ensemble of classifiers which can work directly on incomplete data sets.

Considering all these issues, we choose to adopt CAGE-MetaCombiner (described in detail in [22]), which employs a meta-ensemble model to operate efficiently with missing data, in the framework proposed in this paper. To synthesise, an ensemble is built for each different source of data (data set) by using a distributed genetic programming (GP) tool, used to generate the combiner function, i.e. CellulAr GEnetic programming (CAGE) [23], a distributed/parallel genetic programming implementation. The function selects the more suitable classifiers/models for the specific data sets. The different ensembles perform a weighted vote to decide the correct class.

Each ensemble evolves a function for combining the classifiers, which does not need any extra phase of training on the original data. Therefore, in the case of changes in the data, the function can be recomputed incrementally with a moderate computational effort. The final classification is obtained by computing the error using the same formulae as the Adaboost.M2 algorithm is used by the boosting algorithm by computing the error of the entire ensemble instead of a single classifier as in the original boosting algorithm.

This framework can efficiently work on incomplete data sets when each data set comes from a different source of data and when they are obtained from a partition of an incomplete data set by removing groups of missing features.

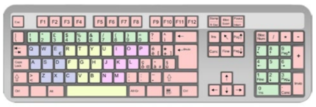
4 Digital footprints and their usage for anomaly detection

This section introduces the anomaly detection method used in this paper and the different sources of log data (digital footprints) chosen as input for this task.

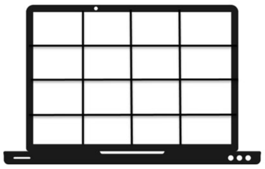
Analysing user behaviour, both for the task of classifying users into homogenous categories and detecting anomalies in their behaviour, requires handling different sources of information, some missing and having heterogeneous characteristics. Indeed, user datas ets can include demographic and education information, e.g. name, age, country, education level, computer knowledge, task knowledge, etc., and may also include information concerning the contest in which the users operate and their roles in the systems. In addition to these data, which usually do not change if we consider a reasonable amount of time, it is necessary to collect operational and behavioural data (e.g. IP addresses from which users connect to the system, operating system and browser used, the duration of the session, etc.), for which changes over time should also be considered. Finally, we collect user input (i.e. commands entered using the keyboard or via GUI, using the mouse, etc.) This information should be captured dynamically by logging user actions.

Unfortunately, all these kinds of data are not present for each user for evident reasons of privacy and for several different motivations (i.e. we have users with different roles, and therefore it is possible to monitor only some types of users, some users

Fig. 1 Mouse and keyboard digital footprints



(a) The division of the keyboard into 5 zones.



(b) The division of the screen into 16 zones.

do not want to give the authorisation to disclose some data). Thus, some sources are missing for different users, and this problem must be faced efficiently to obtain an accurate classification. Usually, all these data are named Digital Footprints, i.e. the traces left by the users while connected to the web, using social networks, or simply by their PCs. More in detail, we monitor three primary data sources: the keyboard, the mouse and the main application/categories in which the user spends most of their time.

As for the keyboard and privacy reasons, as illustrated in Fig. 1a, we only log the zones of the keyboard corresponding to the key the user presses, including alphanumeric characters and special symbols.

As for the mouse, we store all the actions generated by mouse movements and clicks. More specifically, these data refer to the cursor's position on the screen. Still, instead of storing the exact position, we divided the screen into 16 parts (using 4 horizontal and 4 vertical stripes, equally distanced). We save only a number identifying the corresponding screen part where the user clicked or moved the mouse (see Fig. 1b).

The usage of the applications is monitored by the use of CPU, by the number of times the applications are opened and by the usage of memory. The same is made for the categories (to which the applications belong). Finally, all these statistics have been averaged over a time window of 30 minutes.

All the information in the digital footprints left by the users can be opportunely used to discover anomalies in user behaviour. First, we can divide the user into groups based on their behaviour, detected from the ensemble-based classification algorithm, described in detail in Sect. 6.1, which generates models representing the user behaviour. These models permit the definition of a "normal" behaviour inside each group and consequently the determination in a more accurate way of anomalous behaviours.

In practice, the models built are used to identify likely anomalies by associating a class of risk to all the tuples differing by a predefined threshold by the usual behaviour of the user/group. The anomaly detection task is reformulated as a combination of user/group identification tasks, following the principle that the lower the probability that a digital footprint belongs to its corresponding user/group, the

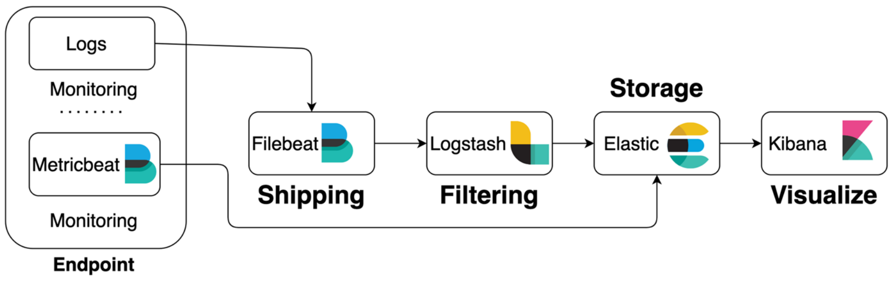


Fig. 2 The elastic search stack

more anomalous behaviour. Given a tuple representing the grouping of the digital footprint coming from different sources in a given time window (30 minutes), we directly predict both which user and group have generated this tuple instead of assessing whether it falls within normal user behaviour and assigns a probability that this behaviour is anomalous proportional to the coverings, the weights and the errors of the different classifiers. In Sect. 6.1, it is described in detail the probability function used to define the anomaly behaviour.

5 Elastic stack: background and scalability issues

In this section, the main features of the elastic search stack are supplied together with the scalability and high-performance features that make this framework apt to our aim, i.e. monitoring and detecting anomalies in large and fast streams of real digital footprints coming from thousands of users.

The elastic stack is a scalable distributed searching system that uses Lucene as the underlying search engine and is composed of three open-source tools, i.e. Elasticsearch, Logstash and Kibana. In addition, some other components, Beats, are used to ship data from different servers and log files to elastic search.

ELK is a powerful tool to search, analyse and visualise data and permits us to cope with the real-time constraints of the logs produced by modern organisations, i.e. GitHub, Netflix, Mozilla, etc.

Figure 2 illustrates a typical usage of the elastic search stack. Some monitoring agents or servers generate the logs, which are sent to the logstash engine through different beats (for example, Metricbeat for the system metrics and Filebeat for log files). Logstash permits filtering data by using a data processing pipeline; it takes data from many sources simultaneously, dynamically transforms them and then sends them to the Elasticsearch processing engine. It is based on some filters, which parse each event and convert them to converge on a standard format for more accessible, accelerated analysis. In this regard, Logstash dynamically transforms and prepares your data regardless of its format or complexity.

Finally, Kibana can be used to visualise the data coming directly from Elasticsearch (or filtered by Logstash) and provide a web-based interface for searching, viewing and analysing these data. It permits the analysis of large volumes of data

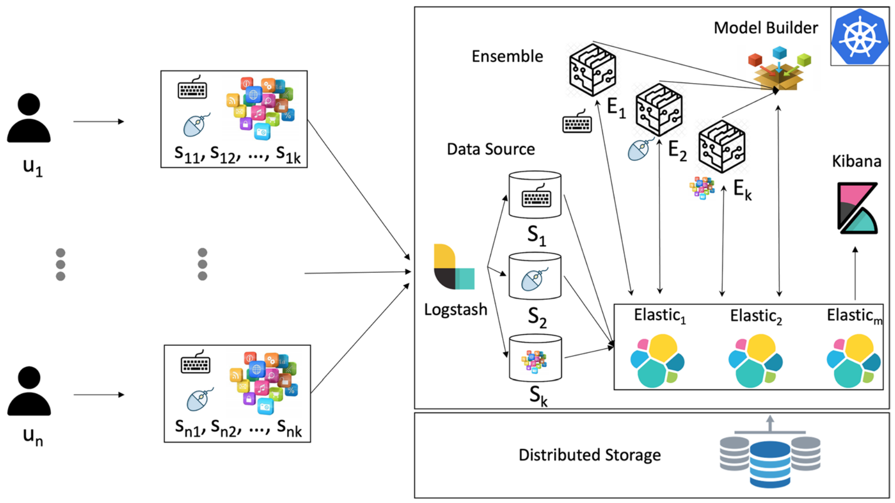


Fig. 3 The software architecture of the Kubernetes-based ELK cluster

and the rapid detection of patterns or irregularities. It allows the quick building of dashboards containing many different bar, line and scatter plots, pie charts and maps.

In practice, elastic is based on a distributed indexing system capable of replicating and routing data into different shards spread across an entire cluster or into the cloud. Each shard maintains a slice of all the data in the index and can be a primary or a replica to provide redundant copies of the data to avoid data loss in case any nodes fail. In our system, shards permit us to split up the indexes horizontally to prevent performance issues and crashes. In addition, a clever mechanism can split index size into shards when the system's performance degrades. In addition, replicas, i.e. copies of index shards, are helpful in backup index data and serving read requests in parallel to improve scalability. The number of replicas can be changed dynamically to boost parallel read performance and resilience. The capacity of the overall system to handle real-time logs is also guaranteed by the nodes composing the Kubernetes cluster, in which the ELK stack is instantiated. The master node can control the Elasticsearch cluster, while the data nodes contain the index and data, and other nodes can be added and work as load balancers. In addition to running on a single Kubernetes-based cluster, Elasticsearch could be deployed on multiple cloud environments and on-premises, using the various automation features available on Kubernetes.

5.1 The ELK-based software architecture of the framework

Figure 3 shows the software architecture of the Kubernetes-based ELK cluster proposed in this work to handle and process the logs coming from many users in real time. On the left of this architecture, several agents (installed on the users' PCs) collect data concerning different sources of information and/or monitoring tools (i.e.

generally automatic software analysing the users' actions and behaviours). These data are collected and usually stored in log files, which are sent to the cluster using Filebeat agents.

On the right side of the figure is illustrated the software architecture running on top of a Kubernetes platform. All the system components are running as Kubernetes services, and thanks to the replica capability of the platform, the system performs well, even when the load varies dynamically. The data coming from the Beats agents are processed by the Logstash service by applying specific filtering and transformation rules. Note that the multi-source logs are gathered by Logstash and opportunely grouped into homogeneous types (i.e. the data concerning the keyboard's output or the process usage of the different users). Then, the data are transmitted to the elastic services that store them on the elastic indexes placed on a Persistent Volume, i.e. a resource descriptor for data storage.

In the Kubernetes platform, the data must be stored on a distributed file system so that every pod (a running container) can access it on every node. The ELK stack is compatible with Kubernetes storage options like EBS, AzureDisk, NFS, Cephfs, GlusterFS, etc. The only requirements are the distributed nature of the data storage and a high-performance capability for the processing requirement of the ELK stack. When these data are stored, the Kibana service and the Model Builder components can access this information for visualisation and computation. Furthermore, given the multi-node nature of Kubernetes, machine learning tasks can benefit from parallel computation: the training of each ensemble can be run in parallel, accessing the data independently.

The overall software architecture is instantiated on a VMWare ESXi virtualisation software cluster. The flexible and user-friendly Kibana online data visualisation platform is designed to serve as an end-point for querying and accessing the tremendous amount of user-mined data gathered through the Elasticsearch indexes. Indeed, the latter is responsible for AI and machine learning tasks and for ingesting and storing large-scale data in an automated manner. In addition, the information extracted from the actual user logs will be held for further analysis, i.e. the classification task and the detection in real time of suspect behaviours and anomalies described better in the following subsection.

6 The main algorithm for classifying user profiles and detecting anomalies

In this section, we show the algorithms used to classify the user profiles in the case of missing data and detect anomalies in user behaviour.

Referring to Fig. 3, we consider the process of monitoring n users, named $u_1, u_2, \dots, u_n$. Each user's digital footprint is monitored by software agents and/or log servers installed on their PC. Logically, the output of each agent constitutes an independent source of data, i.e. for the user i, we obtain the sources $s_{i_1}, s_{i_2}, \dots, s_{i_k}$. All these data sources are sent to Logstash, which collects the information from the different users and filters the information by grouping by data source type and saving the data in the elastic search cluster.

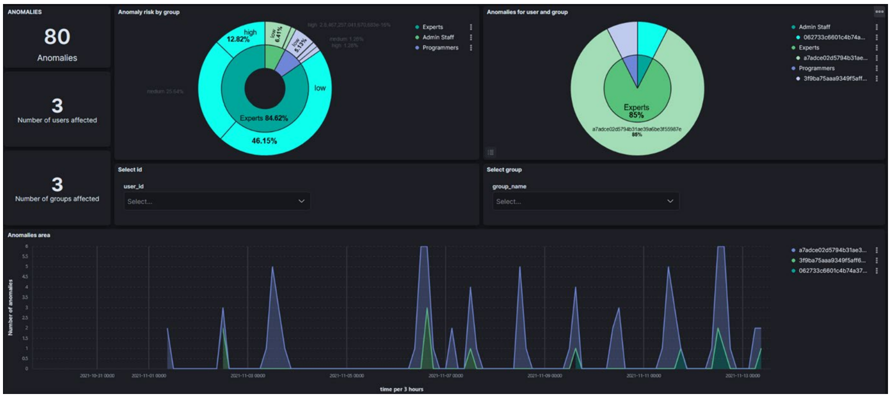


Fig. 4 The Kibana dashboard used for visualising the anomalies

Therefore, the data saved as an index in elastic search, i.e. $S_1, S_2, \ldots, S_k$, are used by the model builder to train the different ensembles, composing the final meta-ensemble, one for each different source of data. We want to specify that the model builder uses only the part of labelled data constituting the training set, as specified later in the pseudo-code. The resulting data stream will be passed to another component able to detect anomalous behaviour for each user/group. The detected anomalies would be visualised by Kibana and/or communicated to the data protection officer, as illustrated in Fig. 4, which shows a dashboard with a typical visualisation of the anomalies, users and groups and the class of risk for a specific time range defined by the data protection officer (DPO).

6.1 Classification of user profiles

In more detail, the classification component is implemented by a modified version of CAGE-MetaCombiner, adapted to work with the users' digital footprints. More formally, the code of this version is detailed in Fig. 5.

In practice, the different sources of data are partitioned into a validation and training data set (respectively $Strain_i$ and $Svalid_i$). Then, a number (α) of base classifiers are trained on the training set, maintaining only the classifiers with a minor classification error. Afterwards, for each data source, an ensemble of the selected classifiers is built using the genetic programming tool to generate the combiner function of the ensemble. Note that no extra computation on the data is necessary, as validation is only used to verify whether the correct class is assigned and compute the fitness function.

A weight, proportional to the error on the training set, is associated with each ensemble and the support for each class, i.e. the decision support matrix is built. This phase could be computationally expensive, but it is performed in parallel, as the different algorithms are independent. The final classification is obtained by computing the error using the same formulae as the Adaboost.M2 algorithm is used by

```
Let $\alpha$ be the total number of base classification algorithms.
Let $\l$ be the number of base classification algorithms effectively used.
Given a set of $k$ incomplete datasets $S_1, S_2, \ldots, S_k$, having $m_1, m_2, \ldots, m_k$ tuples,
where the dataset $D_i = \{X_{i_1}, X_{i_2}, \ldots, X_{i_{mi}}\}$ and $X_{i_k}$ is a tuple $\{A_1, A_2, \ldots, A_d, C\}$
where $A_i$ is an attribute and the class C can have $c$ possible values.
Note that each tuple $X_{i_k}$ can potentially be missing.
**For each** $S_i$
    Consider the dataset $S_i$ partitioned into train and validation : $Strain_i$ and $Svalid_i$
    Train $\alpha$ different classification algorithms on $Strain_i$
    Maintain the $l$ classifiers obtaining the highest accuracy on the training set.
    Build $l$ decision profile matrices, one for each of the classifiers
    using the resp. validation set, one for each classifier of dimension $|Svalid_i| \times c$
    Run the distributed GP tool on the validation set $Svalid_i$ in order to obtain
    the combiner function of the ensemble.
    Obtain an Ensemble $E_i$, a combiner function $F_i$ and a weight $W_i$,
    computed on the basis of the error of the given ensemble on the validation set.
**end for each**
Build the decision profile matrix $DP$ of the entire ensemble $E$,
where each element $H_{i,j}(x)$ is the support that classifier $h_i$ gives to the hypothesis
that the tuple $x$ comes from class $j$.
Compute the weighted mean for each class $j$: $\mu_j(x) = \frac{\sum w_i * H_{ij}(x)}{\sum w_i}$ on the test set.
Compute the class by using the formula $class(x) = argmax_j(\mu_j(x))$.
Note that if a tuple $X_{i_k}$ is missing, the corresponding ensemble $E_i$ does not participate
to the evaluation procedure for that tuple.
```

Fig. 5 The pseudo-code of the distributed classification algorithm

the boosting algorithm by computing the error of the entire ensemble instead of a single classifier as in the original boosting algorithm.

In our case, the different footprint data sets could be incomplete. We assume that each corresponding tuple of the different data sources is used to predict the same class, i.e. the users do not change during the time. Under these assumptions, we can easily handle cases in which a corresponding tuple can be missing in one or more data sets. Still, if it is missing in all the data sets, it will be discarded and counted as a wrong prediction in the evaluation phase.

6.1.1 Detecting anomalies in user behaviour

The classification model (i.e. the metaensemble), defined in the previous subsection, divides the users into homogeneous groups based on their digital footprint. The meta-model built is used to classify each new tuple representing the user's digital footprint and assign a probability of risk representing the possible difference with the usual behaviour of the user/group.

In practice, we reformulated the anomaly detection task as a combination of user/group identification tasks, following the principle that the lower the probability that a digital footprint belongs to its corresponding user/group, the more anomalous the behaviour. Obviously, as we are coping with missing data sources, the probability of recognising an anomaly/normal behaviour is affected /decreased) by the presence of missing groups of features.

In practice, given a tuple representing the overall digital footprint coming from different sources in a given time window (30 minutes in our experiments), we directly predict both which user and group have generated this tuple for assessing whether it falls within normal user behaviour; then, we assign a probability that this behaviour is anomalous using a function based on the coverings (i.e. the ensembles, which participate to the classification of that tuple), the weights and the errors of the different ensembles composing the overall meta-ensemble. The same procedure is repeated for the group, and the overall probability is the weighted average of these two probabilities. The behaviour is considered an anomaly when this probability overcomes a given threshold (0.3 for our experiments).

Now, let us formalise and describe the method used to predict the anomalies, which also works in the case of missing data in the following.

6.1.2 Formalisation of the anomaly detection algorithm

Let $S_{Ens} = \{E_1, \dots, E_k\}$ be the set of the k ensembles composing the meta ensemble, each one having associated a weight w_i, working on a source of data s_i, and that predicts the class among the different class labels $\Omega = \{\omega_1, \dots, \omega_c\}$, representing each one a group (or a user) and c is the total number of classes.

Let $H_{ij}(x) = P(\omega_j \mid x, E_i)$ be the probability that the tuple x belongs to the class ω_j, given by the Ensemble i.

Consider the predicted class $pred_{class}(x) = argmax_j(\mu_j(x))$ for $j \in \{1, \dots, c\}$, where $\mu_j(x) = P(\omega_j \mid x, E) = \frac{\sum_{i=1}^{k} w_i P(\omega_j|x,E_i)}{\sum w_i} = \frac{\sum w_i * H_{ij}(x)}{\sum w_i}$, i.e. the average weighted of all the ensemble probabilities that the tuple x belongs to the class j (with label ω_j).

Define $pred_{class}$ and $real_{class}$, respectively, as the predicted and real class of the tuple. Note that, for the anomaly task aim, the class is the group of the user, which we already know; however, the classification task permits us to estimate the probability that this group be an anomaly, i.e. the tuple is an anomaly.

Then, for each tuple x, we compute the set of the covering $S_{cover} \in \{1, \dots, k\}$ and the set of missing $S_{missing} \in \{1, \dots, k\}$, respectively, as the subset of the indices corresponding to the ensembles that participate in the classification of the tuple x and the set of the ensembles working on the missing sources of data (and therefore not participating on the classification of this particular tuple). α is a constant of reduction, usually with a value slightly minor than 1, useful to add a penalty factor for the ensembles of the covering set.

Afterwards, we compute the probability of anomaly as:

$$P_{Anomaly} = \alpha \frac{\sum w_i * S_{cover} * P(\omega_j \mid x, E_i)}{\sum S_{cover} * w_i} * missing\_factor$$

where *missing_factor* is a factor of reduction of the probability of anomaly in case of missing data. The intuition behind this factor is that it aims to reduce the probability in case of missing data; still, it must guarantee a sufficient value of the probability

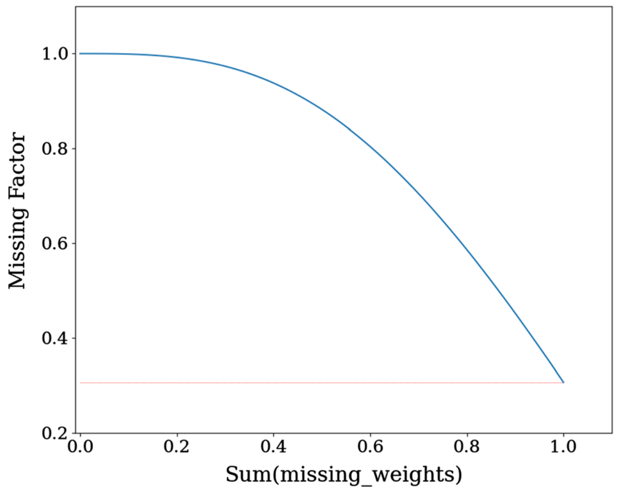


Fig. 6 The log function used to scale the reduction factor for the missing data

of anomaly, even when many sources of data are missing. Therefore, we adopted the following logarithm function:

$missing\_factor = \beta[1 - log(1 + sum\_missing\_weights^3)]$ and $sum\_missing\_weights = \sum_{i \in S_{missing}} w_i$ where β is a reduction factor slightly minor than 1. This function permits the assignment of a reasonable probability of anomaly, even when the percentage of missing data is close to 99%. Indeed, in this case, the value of the function is close to 0.3, considering $\beta = 1$ (that can be further reduced by using the β constant). Therefore, the probability of anomaly was substantially reduced, but we can assign a probability, even when most of the data are missing. On the contrary, when the percentage of missing data is up to 0.1(10%), this reduction factor is almost neglectable, i.e. it is about 0.99; see Fig. 6.

7 Experimental results

In this section, the experiments conducted to analyse the capacity of our approach to handle missing features and operate with real data are described together with the main characteristics of the data sets used.

7.1 Data sets, measures and parameters

This subsection describes the real data sets used to validate our anomaly detection algorithm and the parameters, and the performance measures used.

Table 1 Description and data sources for the HFUser data set

Records	Features	Users (Groups)
2220	85 + 3	10 (3)
Data sources		
DS-1	DS-2	DS-3
Activity keyboard (5 features)	Mouse movement (16 features)	Cpu usage (16 features)
	Mouse click (16 features)	Memory usage (16 features)
	Mouse zone (16 features)	

Three features are shared by all the data sources: user id, timestamp and group id

7.1.1 Real data sets of user behaviour

To validate our algorithm, data sets should have some common desirable properties, not easy to find in literature, listed in the following:

a) data must originate from logs (or other streams) monitoring the behaviour of the users;
b) statistics on the users and logging data can be present together;
c) the users can be divided into homogeneous groups;
d) each data source can be affected by missing data for some users and some observation windows.

Therefore, in addition to the data set used for the classification task and described in Sect. 7.2, to evaluate the goodness of our algorithm of anomaly detection, we used two real data sets, which present the features mentioned above and include both personal data and the result of preprocessing logs resulting from monitoring their behaviour.

The first data set, named *HFUsers*, was supplied by the HFactor Security company and contains the data concerning 10 real employers of a company, divided into three different groups, respectively, named Admin Staff, IT Staff and Experts (experts in the application domain of the company, but not in information technology). The subdivision into groups facilitates the detection of anomalies as it is more probable that homogeneous groups of people exhibit the same behaviour. The behaviour of these users is saved into logs, as illustrated in Fig. 3 for about four weeks. For our experiments, the first week is used as the training set, while the other weeks are used to evaluate the anomaly detection performance of our system. This data set comprises 2, 220 tuples, coming from the three different data sources (i.e. keyboard, mouse and applications) described in Sect. 4. The overall number of features is 88 and the statistics over the log were sampled every 30 minutes. For the previously mentioned reasons, there are many missing data for each sample, as data originating from logs of mouse or keyboard or (less frequently) apps are missing. The data set is partitioned into three sets, and they are used as independent data sources to train the classification models. The list of the features for each data source is reported in Table 1. Each data source contains the references to the user (i.e. the

Table 2 Description and data sources for the ISA data set

Records	Features	Users (Groups)
548531	29 + 3	120 (4)
Data sources		
DS-1	DS-2	DS-3
The questionnaires about browsing behaviours (15 features)	User descriptive fields (6 features)	Indicators about network traffic (8 features)

Three features are shared by all the data sources: user id, timestamp and time slot

id and the group) and the timestamp of the event. The first data source holds the data about the keyboard events registered, while the user is working. The second data source contains the event related to the mouse movements. Finally, the third data source stores the data about the applications running in the user machine. For each application, the agent collects the 30-minute average statistics about memory and CPU usage categorised per application category (i.e. gaming, utilities, development, etc.).

The second data set is the *ISA* context-aware data set described in detail in [24], which contains the information security awareness (ISA) scores assessed from the three data sources (namely, the questionnaires, mobile agent and network traffic monitor) by conducting a long-term user study involving 162 smartphone users, downloadable from here [2]. Considering a time window, a tuple of the data set can be regarded as composed of three sets of features, i.e. the ISA indicators, the subject statistics and Internet traffic. The original data have been collected from three data sources: sensor readings from a mobile device agent installed on the subjects' smartphones; Internet traffic collected using a VPN client installed on the subjects' smartphones, and demographic attributes and self-reported behaviour collected using a security questionnaire. The data set is composed of 548, 531 tuples and 32 features. Table 2 reports the description and the features of each data source for the ISA data set.

We consider missing data for the three horizontal partitions described above, corresponding to the three data sources, as follows. For the ISA indicators, we consider as missing a tuple having no ISA indicator equal to 1 (no factor of risk is present), while for the other two partitions, as better specified in Sect. 7.3, we randomly selected a percentage of missing data varying from 5% to 30%. The faculty attribute permits the division of the users into four homogeneous classes (Natural Sciences, Engineering, Humanities, Others); therefore, we employ this feature to build the different models to classify the users necessary for our anomaly detection algorithm.

[2] https://bit.ly/3HTCj5i.

7.1.2 Parameters and evaluation measures

All the experiments were performed using the elastic stack version 7.15 on a ProLiant cluster running VMWare ESXi with 24 CPUs Intel Xeon E5-2673 2.40 GHz and overall main memory of 128 GBytes. The master node of the cluster is configured with 4 GBytes of main memory, while the slaves are configured with 16 GBytes. No tuning phase has been conducted for the GP algorithm. Still, the same parameters used in the original paper [25] were listed: a probability of crossover of 0.7 and mutation of 0.1, a maximum depth of 7 and a population of 132 individuals per node. All the results were obtained by averaging 30 runs.

Standard metrics for evaluating anomaly detection systems are recall (or true-negative rate) and precision, which give an idea of its capability to individuate the anomalies and to avoid false alarms, respectively. In more detail, recall is the fraction of anomaly behaviour that was reckoned as such (a value of 100% for a model means that it detected all the anomalies, but it might also yield false alarms). In contrast, precision is the proportion of anomaly tuples detected correctly relative to the number of all the tuples recognised as anomalies (a value of 100% means no false alarms were raised, but some anomalies may be missed).

These two measures are usually merged into a single one, named *F-measure*, computed as their harmonic mean and offering a summarised performance score. For conciseness and readability, we will next show the F-measure scores only. Notice, however, that F-measure is not suitable for class-imbalanced data. For this reason, we resorted to two different metrics, namely *AUC* (area under the curve) and *AUC-PR* (area under the precision-recall curve), which have been widely employed in the case of imbalanced data, as described below.

The *AUC* metric quantifies the area under the ROC curve. The ROC curve is computed by comparing the false-positive rate (i.e. the ratio between the number of false alarms signalled and that of all the normal data processed) and the true-positive rate (i.e. the recall).

For imbalanced classes, it was proposed to employ the *AUC-PR* metrics (i.e. the area under the precision-recall curve), which does not rely on the true-positive rate and it is hence less risky to overestimate a model in settings where the number of normal data is sensibly higher than the anomalies. However, because of this aspect, AUC-PR is more benevolent in approaches raising many false alarms. For this reason, these two latter metrics have been used in our experimental analysis.

7.2 Classification results

This work is focused on the task of anomaly detection in monitoring the behaviour of the users. However, to make this work self-contained, in this subsection, we report the classification results already shown in our previous work [8].

In addition to the two data sets of the previous subsection, for the classification task, we use the UNIX data set [26], containing logs of commands of users operating on a Linux shell. These users are profiled according to their expertise with the shell commands. A high number of features makes this data set valuable for testing

Table 3 Classification accuracy (per cent) for Cage-MetaCombiner with the Unix data set for different percentages of missing data

Commands	Percentage of missing		
	0%	20%	40%
100	85.06 ± 0.53	84.26 ± 1.29	83.09 ± 2.01
500	83.76 ± 1.01	83.17 ± 2.12	83.10 ± 2.11

Table 4 Comparison of the Cage-MetaCombiner vs the EvABCD algorithm for the Unix data set (classification accuracy)

Commands	Cage-combiner	EvABCD
100	85.06	64.30
500	83.76	70.80

the algorithm's performance in the case of missing features. The data set is preprocessed in the same way used in [18], i.e. for each user, the first 100 and 500 commands used are considered; then, the commands subsequences of fixed length (from 3 to 6) are extracted from the list. Each user represents a record in the processed data set; all the distinct subsequences are used as record attributes, and the attribute value is the number of times the subsequence is typed by the user (0 means that the subsequence is never typed).

As we are interested in handling cases in which entire groups of missing features are missing and not in coping with random patterns, we simulate missing data by eliminating tuples according to a probability threshold, i.e. this parameter controls the percentage of tuples with missing attributes. The features of the data sets were vertically divided into 4 partitions. Afterwards, if the threshold of missing data is set to 20%, the entire partition of the features belonging to this tuple has a probability of 0.2 to be missing. If all the partitions of a tuple are missing, this tuple will be considered an error of classification. The percentage of tuples with missing data varies from 0% (no missing data) to 40%, and the results of these experiments are reported in Table 3.

Our approach works quite well, even in case of high percentages of missing data and, for the case of an increased number of commands (500), the loss in accuracy, even considering 40% of missing data, remains under 1%. Finally, Table 4 shows the comparison between our approach and EvABCD [18], which first operates with this data. The EvABCD algorithm is a technique for classifying the behaviour profiles, which builds one or more prototypes for each user profile, similar to our approach. However, it is based on the frequency of all the sequences of commands with a defined length and incrementally updates the models. Our algorithm performs better than EvABCD for the 100 and 500 command cases. In addition, while the different number of commands extracted seems not to influence the accuracy of our approach, the EvABCD algorithm works better with a more significant number of commands.

Table 5 Comparison of our approach with different state-of-the-art classification algorithms: F-measure, AUC and AUC-PR for the ISA data set

Algorithm	AUC	AUC_PR	F1-score
CAGE-MetaComb.	0.9977 ± 0.00007	0.9900 ± 0.00051	0.9526 ± 0.00024
ExtraTreesClassifier	0.9952 ± 0.00003	0.9863 ± 0.00010	0.9402 ± 0.00016
KNeighborsClassifier	0.9896 ± 0.00007	0.9617 ± 0.00023	0.9046 ± 0.00032
GradientBoostingClassifier	0.9798 ± 0.00012	0.9369 ± 0.00029	0.8504 ± 0.00055

Table 6 Comparison of our approach with different state-of-the-art classification algorithms: F-measure, AUC and AUC-PR for the HFUsers data set

Algorithm	AUC	AUC_PR	F1-score
CAGE-MetaComb.	0.9994 ± 0.0009	0.9907 ± 0.0137	0.9405 ± 0.0253
ExtraTreesClassifier	0.9988 ± 0.0009	0.9853 ± 0.0179	0.9192 ± 0.0405
KNeighborsClassifier	0.9989 ± 0.0010	0.9774 ± 0.0192	0.9137 ± 0.0239
GradientBoostingClassifier	0.9922 ± 0.0083	0.9225 ± 0.0482	0.8093 ± 0.0555

7.3 Anomaly detection: the masquerade detection scenario

To assess the goodness of our algorithm in detecting anomalies in the behaviour of the users, we adopted a masquerade detection scenario. This scenario considers the case in which some users mimic the behaviour of normal users in order to access reserved information/resources or conduct internal attacks on a system/network. Unfortunately, in general, and for the two data sets described in Sect. 7.1, it is hard to have a ground truth, i.e. a large number of anomalies to evaluate an anomaly detection algorithm. Practically, it is unrealistic to suppose that it is possible to validate the technique with 100% real and sure anomalies.

Therefore, we adopt a masquerade detection scenario under the realistic assumption that some user's activities X should be considered anomalous behaviour when conducted by another user Y. In this regard, we introduce synthetic masquerades by randomly changing the user label for a given time window to obtain an intrusion rate of about 20%. Both the time window and the number of users were chosen randomly; a minimum window of 6 hours was determined to make the scenario realistic and the number of masquerade users was selected in order to reach the desired percentage of 20% of the overall number of tuples. As, respectively, the first week of the data (in the case of the HFUser data set) and the first 25% of the tuple for the ISA data set were chosen for the training set, no masquerade was introduced in this part of the data set.

7.4 Comparison with state-of-the-art algorithms and handling missing data

In this subsection, we compare our approach with other well-known classification algorithms, most of them based on the ensemble paradigm, and evaluate the capacity of our system to handle missing data.

Table 7 AUC, AUC-PR and F-measure of our approach for the ISA data set for different percentages of missing data

Perc.	AUC	AUC-PR	F1-score
5%	0.9977 ± 0.0001	0.9900 ± 0.0005	0.9526 ± 0.0002
10%	0.9971 ± 0.0004	0.9876 ± 0.0024	0.9493 ± 0.0005
20%	0.9959 ± 0.0005	0.9816 ± 0.0036	0.9412 ± 0.0015
30%	0.9949 ± 0.0007	0.9759 ± 0.0049	0.9367 ± 0.0026

Table 8 AUC, AUC-PR and F-measure of our approach for the HFUsers data set for different percentages of missing data

Perc.	AUC	AUC_PR	F1-score
5%	0.9994 ± 0.0009	0.9907 ± 0.0137	0.9405 ± 0.0253
10%	0.9992 ± 0.0011	0.9877 ± 0.0221	0.9346 ± 0.0312
20%	0.9990 ± 0.0012	0.9856 ± 0.0170	0.9285 ± 0.0268
30%	0.9988 ± 0.0014	0.9828 ± 0.0165	0.9196 ± 0.0301

We also tried to compare our algorithm with many known outlier detection algorithms (i.e. LOF, isolation forest, one-class SVM); however, in this particular problem, especially for the considerable number of anomalies present in the masquerade detection scenario, outlier-based approaches do not work well and the AUC obtained in the best performing case among the different outlier algorithms was less than 0.6, far away from the results of our algorithm.

Therefore, we decided to compare our approach with a meta-ensemble of (other) state-of-the-art classification algorithms, equipped with our anomaly detection function. In particular, we choose the Scikit implementation of two ensembles: the ExtraTreesClassifier and gradient boosting classifier and the K-neighbours classifier (also considering the implementation present in Scikit). For a fair comparison, we adopt the same settings of our algorithm, i.e. the same percentage of missing data (5%), the same partition of the features of the data sets for generating the missing data and the same algorithm of anomaly detection, proposed in this paper.

The comparison results are reported in Tables 5 and 6, respectively, for the ISA and the HFUser data set, in terms of AUC, AUC-PR and F-measure.

By analysing the table concerning the ISA data set, it is evident that our approach obtains the best results for all the measures, and the differences are substantial, especially for the AUC-PR and F-measure. At the same time, the MetaEnsemble of ExtraTreesClassifier is the competitor, obtaining the best results compared to the others. As for the HFUser data set, the trend is quite similar. However, the differences in terms of AUC are not so substantial, while for the other two measures, our Metaensemble considerably outperforms the other approaches. Finally, the AUC values for the Metaensemble of ExtraTreesClassifier and of the KNeighborsClassifier are similar.

The second suite of experiments aims to evaluate our approach's robustness when there is a lot of missing data. We randomly varied the percentage of missing data of the two data sets to reach a percentage from 5% to 30%, according to the following procedure. For the HFUsers data set, for each tuple, according to the chosen probability (i.e. 0.2 corresponding to 20%), one of the three horizontal partitions (i.e. mouse, keyboard or apps), randomly extracted, was hidden, so simulating missing data. The same procedure was applied to the ISA data set for the three partitions of ISA indicators, i.e. subject statistics and Internet traffic.

The AUC, AUC-PR and F-measure metrics of this comparison for the different percentages of missing data are reported in Tables 7 and 8, respectively, for the ISA and the HFUser data set.

By analysing the tables, even when the percentage of missing data arrives at 30%, the performance degradation of our algorithm in terms of AUC is very low for both data sets. As for the AUC-PR and especially for the F-measure, a slight grade of degradation is present, but it is acceptable.

8 Conclusions and future work

A high-performance framework based on the elastic stack to process and store large and fast streams of data monitoring the behaviour of the users of a company is proposed.

This framework can efficiently handle missing and unbalanced sources of data; in addition, it integrates digital footprints coming from many sources of data incrementally and can be used for different tasks in preventing cybersecurity problems correlated to the human factor, such as the classification of user profiles and risks and the anomaly detection of misuse/abuse user behaviour.

Experimental results demonstrated that the framework effectively handles missing data in the classification task. In addition, compared to existing state-of-the-art solutions for the two real data sets concerning users' behaviour, the system can effectively detect anomalies, used in the masquerade detection scenario conducted, especially in terms of AUC-PR and F-measure. The excellent performance of our method is also confirmed in the case of unbalanced data containing many missing tuples. Future works aim to extend the system with the support and the implementation of the main tasks using Apache Spark Streaming and to evaluate its scalability in a real-world scenario. In addition, the assessment of the framework could be extended to the case of data sources coming from social, mobile and IoT environments.

Author Contributions G. Folino contributed to resources, supervision, conceptualisation, investigation, writing - original draft, writing - review and editing and funding acquisition. C.O. Godano contributed to conceptualisation, supplying and processing real data, methodology, investigation and writing - review and editing. F.S. Pisani contributed to investigation, software, processing real data, experiments and writing - review and editing.

Funding Open access funding provided by Consiglio Nazionale Delle Ricerche (CNR) within the CRUI-CARE Agreement.

Availability of data and materials The UNIX and the ISA data sets for this research are included, respectively, in [26] and in [24]. The HFUsers data set is available on request from the corresponding author, with the permission of the HFactor Security company. The data are not publicly available due to restrictions, as their containing information could compromise the participants' privacy.

Declarations

Conflict of interest The authors declare no conflict of interests.

Ethical approval This declaration is not applicable.

References

1. CERT Australia (2012) Cyber crime and security survey report. Technical report
2. Subrahmanian VS, Ovelgonne M, Dumitras T, Prakash BA (2015) The global cyber-vulnerability report, 1st edn. Springer, NewYork
3. van Zadelhoff M (2016) The biggest cybersecurity threats are inside your company. Digital article - harvard business review
4. Folino G, Sabatino P (2016) Ensemble based collaborative and distributed intrusion detection systems: a survey. J Netw Comput Appl 66(C):1–16
5. Folino G, Guarascio M, Papuzzo G (2019) Exploiting fractal dimension and a distributed evolutionary approach to classify data streams with concept drifts. Appl Soft Comput 75:284–297
6. Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140
7. Freund Y, Shapire R (1996) Experiments with a new boosting algorithm. In: Machine Learning, Proceedings of the Thirteenth International Conference (ICML 96), pp 148–156. Morgan Kaufmann
8. Folino G, Godano CO, Pisani FS (2022) A scalable architecture exploiting elastic stack and meta ensemble of classifiers for profiling user behaviour. In: González-Escribano A, José Daniel García, Torquati M, and Skavhaug (eds) 30th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP 2022, Valladolid, Spain, March 9-11, 2022, pp 189–196. IEEE
9. Hilal W, Gadsden SA, Yawney J (2022) Financial fraud: a review of anomaly detection techniques and recent advances. Expert Syst Appl, 193(C), May 2022
10. Ahmed M, Mahmood AN, Hu J (2016) A survey of network anomaly detection techniques. J Netw Comp Appl 60:19–31
11. Garg R, Upadhyaya K (2006) Profiling users in GUI based systems for masquerade detection. 2006 IEEE information assurance workshop. IEEE Computer Society Press, Washington, DC, pp 48–54

12. Tabia K, Benferhat S (2008) On the use of decision trees as behavioral approaches in intrusion detection. In 2008 Seventh International Conference on Machine Learning and Applications, pp 665–670
13. Pannell G, Ashman H (2010) User modelling for exclusion and anomaly detection: a behavioural intrusion detection system. In: De Bra P, Kobsa A, Chin D (eds) User modeling, adaptation, and personalization. Springer, Berlin Heidelberg, Berlin, Heidelberg, pp 207–218
14. Corney M, Mohay G, Clark A (2011) Detection of anomalies from user profiles generated from system logs. In: Proceedings of the Ninth Australasian Information Security Conference - Vol 116, AISC 11, pp 23-32. Australian Computer Society, Inc
15. Harilal A, Toffalini F, Homoliak I, Castellanos JH, Guarnizo J, Mondal S, Ochoa M (2018) The wolf of SUTD (TWOS): a dataset of malicious insider threat behavior based on a gamified competition. J Wirel Mob Netw Ubiquitous Comput Depend Appl 9(1):54–85
16. Kim J, Park M, Kim H, Cho S, Kang P (2019) Insider threat detection based on user behavior modeling and anomaly detection algorithms. Appl Sci 9(19):4018
17. Schonlau M, Theus M (2000) Detecting masquerades in intrusion detection based on unpopular commands. Inf Process Lett 76(1–2):33–38
18. Iglesias JA, Angelov P, Ledezma A, Sanchis A (2012) Creating evolving user behavior profiles automatically. IEEE Trans Knowl Data Eng 24(5):854–867
19. Prarthana TS, Gangadhar ND (2017) User behaviour anomaly detection in multidimensional data. In: 2017 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), pp 3–10
20. Garchery M (2021) User-centered intrusion detection using heterogeneous data. PhD thesis, Universität Passau
21. Little RJA, Rubin DB (1986) Statistical analysis with missing data. John Wiley & Sons Inc, New York, NY, USA
22. Folino G, Pisani FS (2016) Evolving meta-ensemble of classifiers for handling incomplete and unbalanced datasets in the cyber security domain. Appl Soft Comput 47:179–190
23. Folino G, Pizzuti C, Spezzano G (2001) Cage: A tool for parallel genetic programming applications. In: Miller JF, Tomassini M, Lanzi PL, Ryan C, Tettamanzi AGB, Langdon WB (eds), Proceedings of EuroGP 2001, vol 2038 of *LNCS*, pp 64–73, Lake Como, Italy, 18-20 April 2001. Springer-Verlag
24. Solomon A, Michaelshvili M, Bitton R, Shapira B, Rokach L, Puzis R, Shabtai A (2022) Contextual security awareness: a context-based approach for assessing the security awareness of users. Knowl Based Syst 46:108709
25. Folino G, Pizzuti C, Spezzano G (2003) A scalable cellular implementation of parallel genetic programming. IEEE Trans Evol Comput 7(1):37–53
26. Greenberg S (1988) Using unix: collected traces of 168 users. In: Research Report 88/333/45. Department of Computer Science, University of Calgary, Calgary, Canada