Documentation of RELOAD

RELOAD (Rapid EvaLuation Of Anomaly Detectors) is a tool that allows to easily compare different algorithms for anomaly detection. It is written in Java, wrapping algorithms coming from other Java-based frameworks such as ELKI or WEKA. The tool is not meant to provide an extensive pool of algorithms and to offer optimized detection speed, but instead is shaped as a simple GUI tool that allows also non-experts or practitioners of other domains to use machine learning algorithms and apply them for anomaly detection. The tool itself does not provide novel technologies, algorithms or techniques, but instead adopts state-of-the art findings in an orchestrated and simplified fashion. As opposed to existing tools and web portals as *ELKI*, *WEKA*, *OpenML*, *RapidMiner* and others, RELOAD provides the user with

- *Easy to use*, since it embeds known techniques by hiding many implementation details and variants to the final user, which is requested to select just a few inputs. Indeed, it is a powerful artifact to be used as a teaching support for bachelor or masters' degrees.
- *Open source*, since code is available on online public repositories and does not require the user to pay for the download, the documentation or any monthly/yearly fee to keep using the tool.
- *Lightweight and portable*, given that Java 8+ is installed in the target machine.
- *Easily extensible* by adding new algorithms, with built-in interfaces to embed algorithms and other techniques e.g., feature selection strategies, depending on the needs of the user.
- *Shaped for anomaly detection,* including a full – but almost transparent to the user – support to sliding windows.

The tool at its current state can efficiently process static data sources i.e., datasets that were already created for off-line analyses. Support for data streams for run-time anomaly detection is currently under investigation.

State of the Art

Understanding the relevance of tools to evaluate and compare anomaly detection algorithms is intuitive. In fact, it is generally difficult to perform an extensive experimental campaign without supporting frameworks that automate execution of experiments and data analysis.

For data mining purposes, frameworks such as *ELKI* [2], *WEKA* [5], *RapidMiner* [3], portals as OpenML [37] or libraries such as *Scikit* [4] were created that allow comparing the performance of essentially *any* data mining and machine learning algorithm following a specific methodology. ELKI and WEKA provide Java-based executables with built-in algorithms including anomaly detection ones, while Scikit is a *Phyton* library for data mining that includes anomaly-based techniques such as *Random Forests* [30] or *Gradient Boosting* [7]. Despite these solutions offer the opportunity to extend their functionalities, they require extensive customization to execute algorithms for determined purposes and settings, and in general their full set of functionalities is not intuitive to grasp. Instead, *RapidMiner* [3] is an enterprise suite that offers graphical support with a complete user interface, but extending the pool of algorithms is rather complicated and not all of its functionalities are free of charge.

Summarizing, these powerful solutions already include some anomaly detection algorithms, and new ones can be added. As a drawback, we observe that features, settings and usage are not always easy to understand and do not offer a complete support to the *experimental evaluation of anomaly detection algorithms in the domain of dependable and secure systems*. In particular, the following requirements can be achieved – when possible – with the above solutions only through extensive customization:

- Focus on unsupervised algorithms, which are often acknowledged as the most suitable way to identify unknown vulnerabilities as zero-day attacks [1], [31]: implementations of unsupervised algorithms are scattered in the above-mentioned frameworks (e.g., Isolation Forests [25] can be found only in [5]).
- Provide support for sliding windows algorithms, that are often relevant [18], [8], [32].
- Integrate different feature selection techniques, allowing also to sequentially executing a pool of the above techniques. This way, it is possible to define a subset of relevant indicators to be monitored, and evaluate voting strategies to decide on anomalies on the basis of the scores from such indicators. This is important to maximize detection efficacy and minimize resource consumption.

Characteristics of RELOAD

The peculiarities and characteristics of RELOAD can be described along the following lines.

Domain

RELOAD provides functionalities specific for the evaluation of anomaly detection algorithms intended for attack or failure detection. In fact, extensive support is offered for unsupervised algorithms, sliding window algorithms, automated tailoring of features/indicators, profiling of algorithms parameters, and voting strategies.

Input/output data

Our tool can load data from different kinds of data sources, that may either be i) text files, or ii) MySQL databases, while a support for data streams is currently under development. In addition, it provides outputs results in CSV files that can be easily manipulated. Currently, it contains loaders to the attack data sets KDDCup99 [12], NSL-KDD [9], ADFA-LD [13], ISCX2012 [10], and UNSW-NB15 [11], and to the failure data set available in [33].

Feature Selection

Once datasets are loaded, the user may want and should apply feature selection techniques [16], to filter out indicators that will not provide algorithms with actionable data for the purpose of identifying errors or attacks. While some built-in strategies are customized, e.g., *variance*, *Pearson correlation* [16], a big pool of feature selection strategies as *information gain* stem from the implementations in the WEKA [5] framework.

Algorithms

RELOAD is able to integrate existing algorithms as well as novel algorithms written by RELOAD users. Further, it offers the opportunity to run different algorithms on data sets and collect scores. To facilitate comparison, RELOAD includes 11 algorithms, selected among those already used in research works for failure and intrusion detection. The algorithms are selected from the six main families [1], [15]: *clustering* (K-Means [2]), *statistical* (HBOS [20], SPS [8]), *classification* (SVM [23], Isolation Forest [25]), *neighbour-based* (kNN [22], ODIN [21]), *density-based* (LOF [19], COF [26]) and *angle-based* (ABOD [24], FastABOD [24]). Some of the algorithms were implemented by RELOAD users, while most of them were imported from ELKI, WEKA and RapidMiner. In addition, the tool embeds 6 sliding window algorithms such as SPS [8]. The other 5 sliding windows algorithms are simulated sliding versions of existing algorithms LOF, KNN, ABOD, IsolationForests, KMeans, since no public implementations were made available by authors of sliding window algorithms [38], [39], [40], [41], [42]. Every time a new data point is added to the window, we run the corresponding non-sliding algorithm by using the content of the window as training set. Despite not being optimal in terms of execution time this simulation allows estimating the detection capabilities of non-sliding algorithm when applied to sliding windows.

Metrics

The tool computes the most relevant metrics for evaluating anomaly detectors [6], as they were selected surveying research papers. Built-in metrics are reported and discussed in Section III. This offers a comprehensive evaluation with the default configuration of RELOAD. New metrics can be defined if needed.

Profiling

RELOAD automatically: *i*) operates to identify the most relevant features/indicators of a data set; *ii*) evaluates algorithms with multiple input parameters; and *iii*) checks different voting strategies to decide on an anomaly, given the scores from multiple indicators.

Usability

The tool has an intuitive GUI to select the target algorithms, define configuration parameters, and select the data sets or data streams. It does not need relevant expertise; for example, it was successfully used by students with only limited experience on monitoring and experimental evaluation.

Extensibility

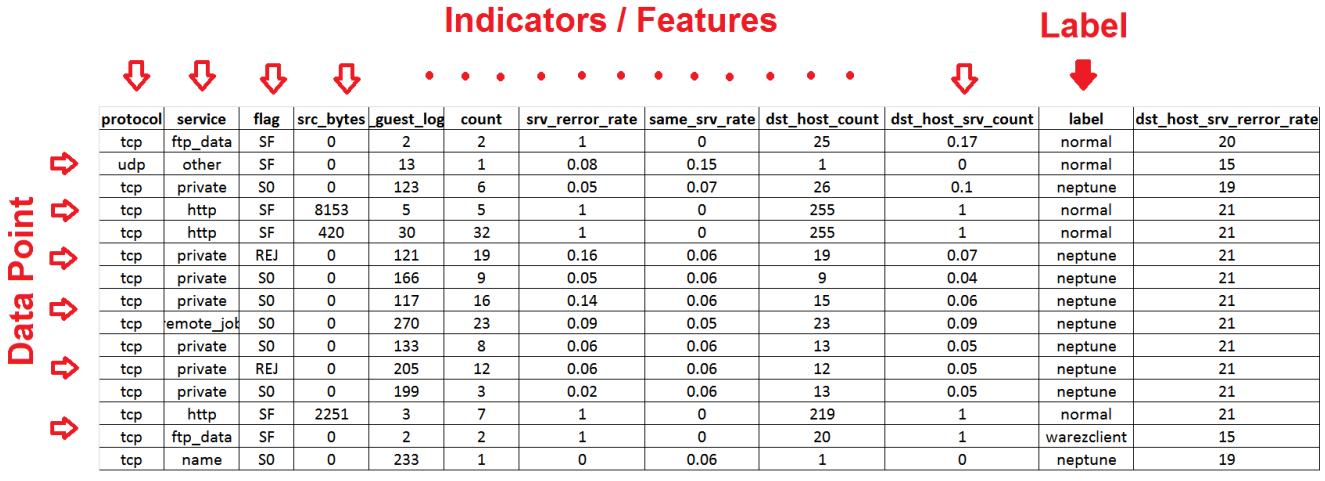
RELOAD can be modified to add new algorithms, metrics or voting strategies. It is open source, developed in *Java*, and available at [17].

Inputs of the Tool

Data set, data streams and indicators

We use the term *indicators* (features) to identify the monitored values that are collected observing the target system. Examples of indicators are the used memory [27], number of cache accesses [29], or number of network packets received in a time interval [28]. We use the term *data point* to refer to the set of values observed, for all the indicators and for a given data set, at a given instant of time. For example, in a log file, usually a data point corresponds to a row (see Figure 1). Noteworthy, we distinguish between training data (TD) and the rest of the data set, that we call evaluation data (ED). Further, we consider only labelled data sets i.e., anomalous data points are explicitly marked. Despite the focus on RELOAD is on unsupervised algorithms, labels are needed to compare the effectiveness of different algorithms or the effectiveness of different parameters' setup for a given algorithm.

Data sets are read through *loaders*, or rather configuration files that contain information about the data sets.



protocol	service	flag	src_bytes	guest_log	count	srv_rerror_rate	same_srv_rate	dst_host_count	dst_host_srv_count	label	dst_host_srv_rerror_rate
tcp	ftp_data	SF	0	2	2	1	0	25	0.17	normal	20
udp	other	SF	0	13	1	0.08	0.15	1	0	normal	15
tcp	private	S0	0	123	6	0.05	0.07	26	0.1	neptune	19
tcp	http	SF	8153	5	5	1	0	255	1	normal	21
tcp	http	SF	420	30	32	1	0	255	1	normal	21
tcp	private	REJ	0	121	19	0.16	0.06	19	0.07	neptune	21
tcp	private	S0	0	166	9	0.05	0.06	9	0.04	neptune	21
tcp	private	S0	0	117	16	0.14	0.06	15	0.06	neptune	21
tcp	emote_job	S0	0	270	23	0.09	0.05	23	0.09	neptune	21
tcp	private	S0	0	133	8	0.06	0.06	13	0.05	neptune	21
tcp	private	REJ	0	205	12	0.06	0.06	12	0.05	neptune	21
tcp	private	S0	0	199	3	0.02	0.06	13	0.05	neptune	21
tcp	http	SF	2251	3	7	1	0	219	1	normal	21
tcp	ftp_data	SF	0	2	2	1	0	20	1	warezclient	15
tcp	name	S0	0	233	1	0	0.06	1	0	neptune	19

Figure 1. Screenshot of the NSL-KDD [9] dataset, showing some indicators/features and some data points.

Through the loaders, RELOAD gathers raw data and metadata from the data set. At the current state of the implementation, default loaders allow connecting to i) CSV, ii) ARFF text files, and ii) MySQL databases. The loaders should specify the data sets type, structures, and the way the dataset is partitioned into TD and ED. RELOAD may also operate with data streams: this requires that the user implements a socket client that listens on the stream and transmits data points in the form of CSV or ARFF strings to the RELOAD server socket. For simplicity, we will only consider data sets in the rest of the discussion.

Feature Selection Strategy(ies)

The user can and should choose one or more strategies for feature selection that he wants to apply when gathering data. At the current state of the implementation, the tool allows selecting features through *Variance*, *Pearson Correlation*, *Information Gain* strategies, while learner-based alternatives as the ones based on Random Forests are currently under development. Note that different feature selection strategies may be applied sequentially by RELOAD.

Algorithms

Currently, RELOAD includes 11 unsupervised anomaly detection algorithms and 6 sliding window algorithm. interfaces are built in the tool, allowing the user to extend existing abstract classes mainly implementing two methods responsible of i) how the algorithm performs its initial training (if empty, as in sliding window algorithms, no training is performed), and ii) calculating the anomaly score the algorithm calculates for a given data point. Since several algorithms were taken from existing frameworks as ELKI and WEKA, wrappers to call and execute such algorithm are already deployed into the tool. When the user chooses which algorithms to execute, he can also propose various configuration parameters of each algorithm (e.g., number *k* of relevant neighbours for neighbour-based algorithms [21], [24]) through a configuration file: RELOAD will replicate the evaluation multiple times, testing all parameters and reporting on individual scores and highlighting the more convenient parameters' combination.

Metrics

RELOAD includes a broad range of metrics to measure the effectiveness of anomaly detectors [6]. Metrics are correct detections (true positives, TP, and true negatives, TN), missed detections (false negatives, FN), wrong detections (false positives, FP), as well as the aggregated metrics [6], precision (P), recall (R), false positive rate (FPR), accuracy (A), F-measure (F1), Matthews coefficient (MCC) and area under ROC curve (AUC). All available metrics are computed whenever RELOAD is executed with its default configuration, while a target metric should be selected by the user to rank algorithms and their effectiveness in detecting anomalies. For example, system requirements may suggest reducing the occurrence of missed detections (FN), even at the cost of a higher rate of FP, favouring metrics such as *recall* instead of *precision*.

Components of the tool

This section reports on the main modules that execute and evaluate anomaly detection algorithms. Such modules are highlighted in Figure 2 by red bold-font labels.

Anomaly Checkers Manager

This component generates and operates a set of *anomaly checkers*: we define an anomaly checker as a unique couple *<indicator, algorithm>*. For each data point, an anomaly checker is able to decide if an anomaly is raised i.e., the anomaly checker produces an anomaly score.

The indicators can also be *composed indicators* [33]. In fact, the Anomaly Checkers Manager operates to aggregate indicators using given relations. For example, two indicators as *bytes sent per second* and *bytes*

received per second may be aggregated using the division operation, creating a composed indicator that defines the *byte sent/byte received rate per second*. Filtered indicators may be aggregated depending on specific rules e.g., always aggregate all the filtered indicators into a unique composed indicator, or if specific properties are met e.g., two filtered indicators are highly or loosely coupled [16]. Strategies to create composed indicators embedded into RELOAD are described later.

Policies Selector

This component evaluates each anomaly checker according to a metric. More in detail, each anomaly checker is applied on a data set (using the Anomaly Checker Manager), and a score is computed for a given metric. Consequently, the Policies Selector can rank the anomaly checkers, and decide on the most effective ones. As *selection policies*, the Policies Selector includes and applies by default the metrics in [33]:

- BEST *x*: the *x* anomaly checkers that have the best ranking according to a specific metric;
- FILTERED *y*: the *y* anomaly checkers with the best ranking, filtered to avoid more than one anomaly checker built using the same indicator;
- THRESHOLD *z*: all the anomaly checkers that reach a threshold *z*, for a given metric e.g., *Recall* > 0.6.

Additional selection policies can be included by updating a configuration file.

Voter

The Voter combines the outputs of a given set of *m* anomaly checkers. The Voter applies a *voting strategy* [14], and identifies a data point as anomalous if at least *n* out of *m* anomaly checkers raise an anomaly. The value *n* can be defined through a configuration file; alternatively, the voter contains the following default voting strategies [33]:

- ALL: the data point is considered anomalous only if all the anomaly checkers identifies an anomaly;
- QUARTER/THIRD/HALF: the data point is considered anomalous only if a quarter/third/half of the *m* anomaly checkers identifies an anomaly;
- ONE: the data point is evaluated as anomalous if at least one of the *m* anomaly checker raises an anomaly.
- <*n*>, where n is the number of checkers that should agree on the anomaly.

GUI

A simple and intuitive UI allows importing inputs and showing outputs; details can be found below.

Methodology Used by the Tool

The workflow of RELOAD is depicted in Figure 2. From left to right, we can observe four phases, namely *initial*,

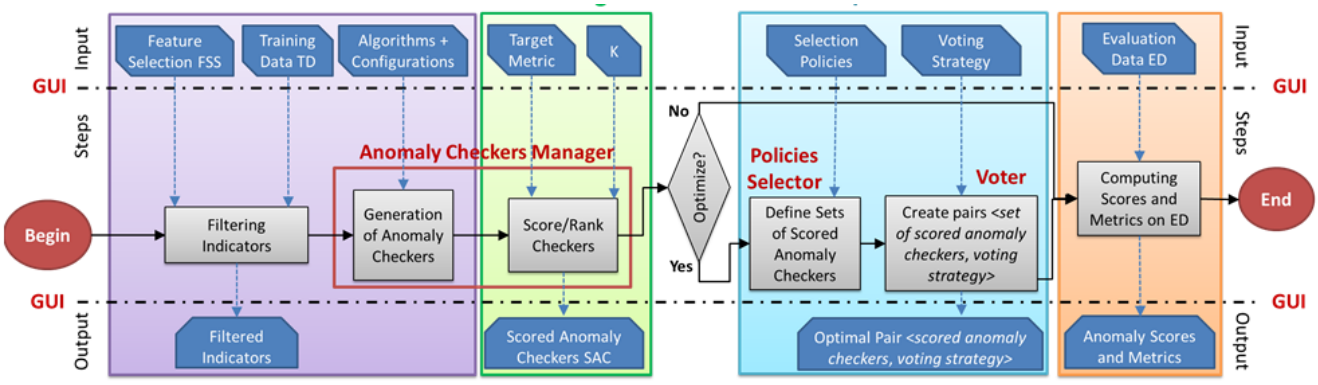


Figure 2. Workflow of the RELOAD tool.

training, *optimization* and *evaluation*. A background process is in charge of automating the execution of the various steps.

Initial Phase

The operations in the *initial phase* are performed only once, when setting up the evaluation. The training data TD is an input of this phase, as well as the feature selection strategies FSS the user wants to apply. The indicators are automatically selected according to FSS, removing those not useful for anomaly detection. Selected indicators may then be combined creating composed indicators; selected indicators and composed indicators build the filtered indicators.

Only the resulting set of *filtered indicators* will be used in the following steps of the methodology. Algorithms are also an input of this phase: anomaly checkers are built using the selected algorithms and the filtered indicators. This phase is mostly realized by the components Anomaly Checkers Manager and GUI.

Training Phase

The *training phase* produces the list of Scored Anomaly Checkers (SAC) using the TD. First, the input metrics are imported. A specific metric, that we call *target metric*, is selected by the user (through the GUI). It is up to the user to understand the target metric that is more relevant for the system under analysis. Then, multiple instances of each checker defined in the initial phase are generated, one for each possible value of algorithm's parameter(s).

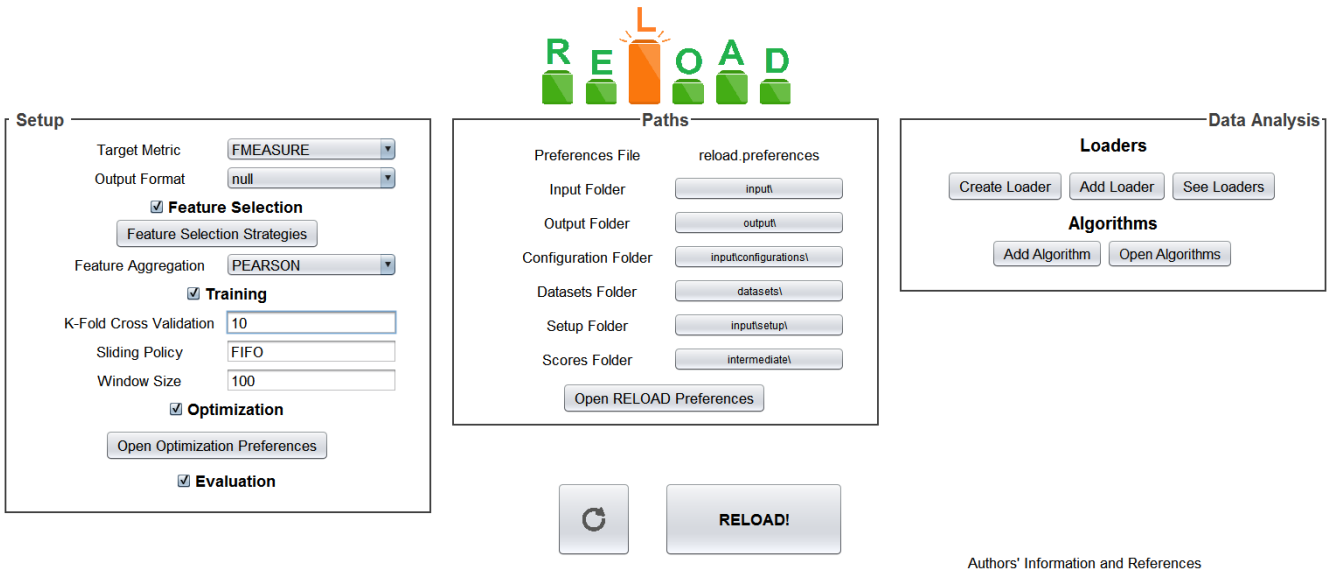
At this stage, RELOAD partitions the TD in two sets: *checkers_train* and *checkers_test,* e.g., 70%-30% split. Then, it uses the *checkers_train* to train the anomaly checkers, and it uses the *checkers_test* as evaluation of the performance of the anomaly checkers, according to the target metric. This allows associating quantitative values to each anomaly checker: the checkers are now *scored anomaly checkers* (SAC) and can be ordered by score.

For example, consider an experiment with two datasets, D1 and D2, using the kNN algorithm with k ϵ {3, 5, 10}. RELOAD generates two checkers, AC1 = <D1, kNN> and AC2 = <D2, kNN>, and each checker is instantiated three times: AC1 = {<D1, kNN(3)>, <D1, kNN(5)>, <D1, kNN(10)>} and AC2 = {<D2, kNN(3)>, <D2, kNN(5)>, <D2, kNN(10)>}. Each instance of kNN is trained using checkers_train; then, instances of each anomaly checker AC1 and AC2 are scored using checkers_test to extract the instance of AC1 and AC2 that guaranteed higher scores according to the target metric.

As a final remark, the user may tune the k parameter in Figure 3 to achieve *k-fold cross validation* [34], which is largely adopted in the machine learning domain to reduce biases derived by inadvertently overfitting the model to the training set.

Optimization Phase

RELOAD users may want to apply combinations of selection policies and voting strategies to understand i) the optimal set of the scored anomaly checkers, and ii) the best *voting strategy* to decide if a data point is anomalous. The *optimization phase* is consequently started. Noteworthy, this phase is optional: it is skipped if RELOAD users are satisfied with executing all the scored anomaly checkers on the ED with no aggregations i.e., having individual results for all the scored anomaly checkers. In the optimization phase, selection policies are applied on the scored anomaly checkers. This allows identifying sets of scored anomaly checkers that satisfy the selection policies. Each of these sets is then matched to the voting strategies. The optimization phase terminates with the definition of pairs <*set of scored anomaly checkers*, *voting strategy*>. This phase is mostly realized thanks to the Policies Selector and the Voter components.



Authors' Information and References

Figure 3. GUI to setup RELOAD.

Evaluation Phase

Last, in the evaluation phase, all the pairs <*set of scored anomaly checkers*, *voting strategy*> are exercised on the ED, to investigate which data points are anomalous. Comparing the outcomes of the evaluation with the labels in the ED, the results are finally computed, for the various metrics. This phase is performed mostly by background processes, with the support of the Anomaly Checkers Manager and the GUI, that visualizes results.

GUI Details

The RELOAD GUI helps setting parameters, selecting algorithms and data sets, and checking results. Here we report details about *i*) *setup* see Figure 3, and *ii*) *results* GUIs (Figure 7, Figure 8). The rest of the RELOAD interaction with the user is limited to configuration files e.g., the loaders, and to output files with results.

Setup GUI

Setup Box

From top to bottom, the user performs the following actions. First, the user selects the target metric. Second, the output format is selected: the default option *null* provides textual information, while the option *GRAPH* creates summarizing graphs. We do not detail on this feature for brevity, but it just creates bar charts with the x-axis containing the data points, and the y-axis reporting the number of anomaly checkers raising an anomaly (a chart is created for each combination of algorithm, data set, voting strategy and selection policy).

Clicking the *Feature Selection Strategies* button opens the window in Figure 4. This GUI describes the available feature selection strategies, allowing the user to choose the ones that suit the problem the most. Selecting more than one strategy leads applying each strategy sequentially according to the order specified by the user in the table.

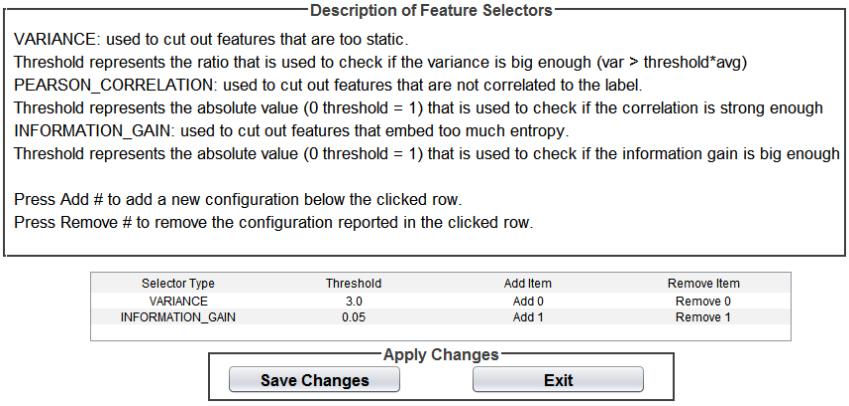


Figure 4. GUI for setup of Feature Selection Strategies.

Then, strategies to aggregate the features/indicators that were selected at the previous step include: i) NONE, that does not combine selected features, ii) UNION, the aggregation of all the selected features as a composed indicator, iii) SIMPLE, that considers selected features individually (as in NONE) and also the UNION indicator, and iv) PEARSON, that considers selected features individually and creates a composed indicator if the *Pearson correlation index* between two or more features/indicators exceeds a given threshold that can be set by the user.

Training phase heavily relies on the setup of the k parameter for k-fold validation [34]. If the analysis targets sliding window algorithms, the size of the sliding window should be set (field *Window Size*). Moreover, the *Sliding Policy* field explains how to manage the sliding window, or rather how to decide if we want to add a novel data point and which data point in the window should be discarded. At the moment, two options are implemented: *FIFO*, the window always slides, replacing the oldest data point with the current one data set is read from top to bottom, and *FIFO_Normal*, that allows the window to slide as FIFO only if the current data point is evaluated as normal by the algorithm. This last strategy avoids polluting the sliding window – which is used by algorithms to learn the expected behaviour of the system – with anomalous data points.

Lastly, the selection policies and voting strategies are configured by opening a configuration file with the button *Open Optimization Preferences*: this file enlists the policies to apply e.g., BEST 3, FILTERED 10, along with the voting strategies e.g., ALL, HALF.

Paths Box

First, the user defines the path of the *input* folder, that is the root path of all the configuration files and folders described below. Then, the user selects: *i*) the *output* folder, which will contain all the results in CSV format; *ii*) the *configuration* folder, which contains the loaders and the metrics configuration files; iii) *datasets* folder, that specifies where textual files (if any) of datasets are placed, and *iv*) the *setup* folder, that contains the data sets. Then, the *score* folder is identified: it is used only for temporary storage of ranked anomaly checkers during computations.

Data Analysis Box

The user selects the algorithms, among the implemented ones, and the data sets, among those in the *setup* folder and that have a loader. When a new data loader has to be defined, as it is the case pressing the “Create Loader” button in the “Data Analysis” box allows to choose a file name and opens a GUI to specify key items of loaders. Depending on the type of the loader e.g., file, database, RELOAD allows the user to choose relevant items to extract data from the chosen data source.

Once the data loader is defined, the user can choose the algorithms he wants to apply on such dataset(s). By clicking the “Add Algorithm” button, RELOAD opens a small window that allows choosing amongst all the available algorithms. Multiple algorithms, and combinations of two or more algorithm, may be selected through this GUI showed in Figure 5. Note that if one or more sliding window algorithms are selected, RELOAD will examine such sliding window algorithm(s) considering

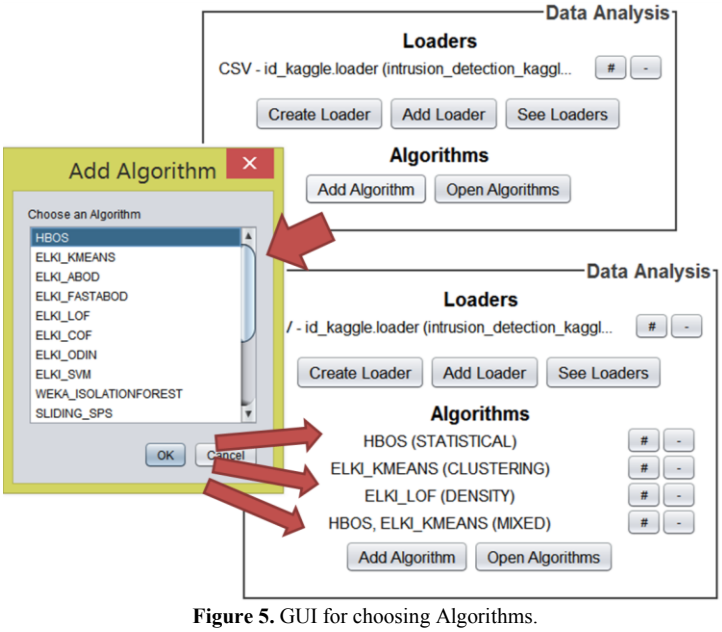


Figure 5. GUI for choosing Algorithms.

the *sliding policy* and *window size* values that the user can set through the GUI showed in Figure 4, box “Setup”.

On the bottom of the GUI, the *Update* button is used to refresh the interface whenever a configuration is modified, while the *Run* button starts the experiments.

Summary and Outputs GUI

RELOAD provides a main summary GUI, while producing many files that expand on specific aspects. More in detail, RELOAD creates, for each <dataset, algorithm> couple, files that report on i) the selected features, and the scores they reached on each feature/indicator selection strategy, ii) the combined indicators created, iii) the ranked anomaly checkers, the optimal voting strategy and anomaly threshold, if optimization is executed, and iv) a detailed list of the anomaly evaluations provided by each anomaly checker used for evaluation, along with the anomaly evaluation generated by the ensemble of checkers by using voting strategy and anomaly threshold. In addition, RELOAD creates detailed views of each combination of data loader and algorithm the user selected for the experiment. This detailed view shows metric scores of the algorithm on a given dataset by varying selection policy and voting strategy, both for training (and, when selected, optimization) and evaluation phases. Screenshots of the two GUIs can be found in the running example.

Downloading the Tool

RELOAD can be downloaded as a ZIP archive from [17]. The ZIP archive includes three items: the JAR file of RELOAD, a preferences file and a folder that contains configuration files. Once files are extracted from the archive, RELOAD can be launched from command line as *java -jar RELOAD.jar*. Note that *Windows OS* users may run the tool by double-clicking the JAR file: this does not show the log of the application (which, however, is saved as a txt file for post-mortem analyses).

Overall, RELOAD needs just a *reload.preferences* file to start, along with the input folder that can be found in the ZIP file. The folder contains main configuration files, which can be managed and adjusted through RELOAD GUI and that are required by algorithms to execute.

System Requirements

RELOAD requires *Java 8* or higher, around 30MB of storage space plus the space needed to store the targeted dataset(s). In this case, since we want to download our dataset and the tool from their online repositories [36], [17], we also need internet connection. We remark here that, once downloaded, the tool is fully standalone and does not require external connections.

Tool Usage: Running Example

We applied previous and less-stable versions of RELOAD to different case studies, especially regarding data logs of service-oriented systems [33]. The tool turned out to be helpful in identifying the more fitting algorithms, either for error or intrusion detection. To show the steps that a generic user has to follow for using the tool, in this section we refer to an entirely new case study targeting the most relevant dataset obtained by querying “intrusion detection” in the *Kaggle* datasets portal. Note that results of the application of RELOAD on some of the other datasets showed by *Kaggle* i.e., *KDDCup* and *UNSW*, may be found at [35].

Analysing and Refining Dataset

After downloading the ZIP file of the dataset, we observe that the dataset is partitioned in two (CSV) files, one for training and the other one for testing. The structure of the two files is the same, expect for an unlabelled

column in the training set – the first one – that we remove due to the lack of information. The resulting data is structured in 42 columns, with respectively 125.973 and 10.000 data points for train and test set. The last column shows the label for each data point, that can be either *normal* (43.3% of the test set), or representing an attack *dos* (33.3%), *probe* (10.5%), *r2l* (12.0%), or *u2r* (0.9%). Information on the attacks are not reported in the portal, we assume that they cover the same categories as their KDDCup [12] and NSL-KDD [9] datasets. All but columns 2 and 42 (1 and 41 if zero-based numbering) are numeric, meaning that RELOAD can process them without needing further categorizations.

Configuring RELOAD

Once started, RELOAD shows the GUI in Figure 3.

Setup Box

Starting from the "Setup" box, we first want to define the reference metric e.g., F-Measure in Figure 3) and strategies to select indicators/features and aggregate them to create composed indicators. For this case study we selected VARIANCE(3) and INFORMATION_GAIN(0.05) feature selection strategies. The other options of the "Setup" box allow choosing how to create composed indicators, which for this case study is executed whenever the *Pearson correlation index* between two or more selected indicators is more than 0.8 – PEARSON(0.8) -, and to choose which of the phases in Figure 3 the user wants to execute. To the sake of this case study, we will run all the phases of RELOAD, thus checking all the *Feature Selection*, *Training*, *Optimization* and *Evaluation* checkboxes. We also proceed with a 10-fold sampling of the training set as widely suggested [34] in the literature.

Path Box

The "Path" box should not need further adaptations. Only note that the default folder for datasets is specified as a "datasets" subfolder of the current directory. If the datasets the user wants to analyse is located in another folder, the user should either i) change the default path of RELOAD through GUI, or ii) move the files.

Data Analysis Box

Here the user can i) define the data loader(s), and ii) select algorithms.

When a new data loader has to be defined, as it is the case for this *kaggle* dataset, pressing the "Create Loader" button in the "Data Analysis" box allows to choose a file name and opens the GUI in Figure 6. We filled the fields of the *id_kaggle.loader* loader in the figure as follows. The CSV files were put in *datasets/intrusion_detection_*kaggle folder: we specified train and test file in the TRAIN_CSV_FILE and VALIDATION_CSV_FILE items. Then, we chose to analyse the performances of RELOAD in identifying probe attacks in this dataset. Therefore, we FAULTY_TAGS fields to "probe" and the SKIP_ROWS fields to "dos", "u2r", "r2l", or rather the remaining attacks we are not interested in. Note that labels are used in the training phase to rank anomaly checkers rather than being provided to algorithms, which work in an unsupervised fashion. We select 50 batches both

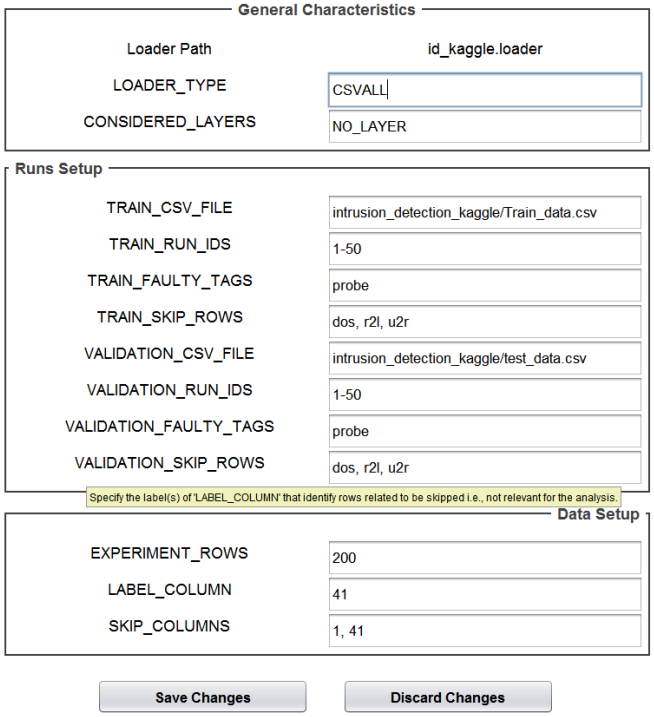


Figure 6. GUI for setting up Loaders.

for training and for validation i.e., RUN_IDS fields, considering batches of 200 data points, as specified by EXPERIMENT_ROWS. Lastly, we specified the LABEL_COLUMN containing labels in zero-based numbering, and the columns to be skipped (see SKIP_COLUMNS field in Figure 7).

To show the versatility of the tool, for this example we selected different algorithms as *HBOS*, *KMeans* and *LOF*, a sliding window algorithm (*SPS*), and a combination of two notoriously fast algorithms such as HBOS and KMeans.

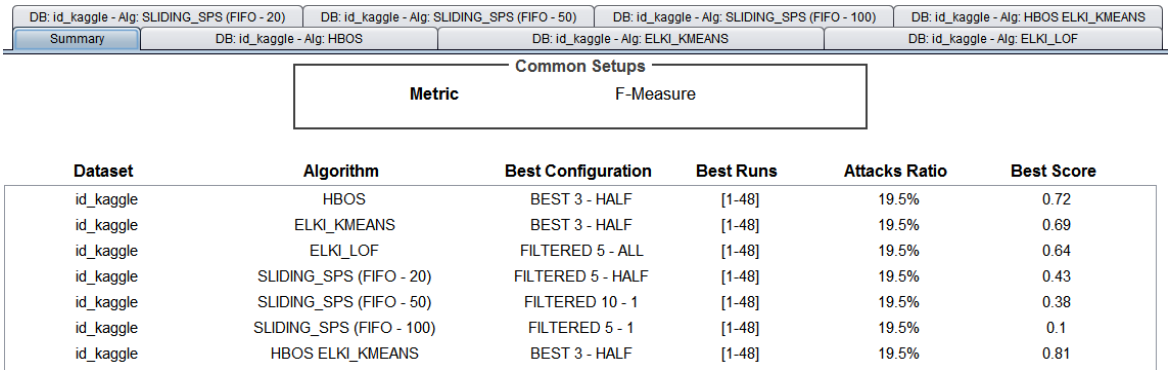
Running RELOAD

When everything is set, the user should press the RELOAD! button on the bottom of the GUI. This will start the process of selecting algorithms and executing anomaly detection. When the process completes, RELOAD will open a window that summarizes results, as shown in Figure 7.

RELOAD Outputs

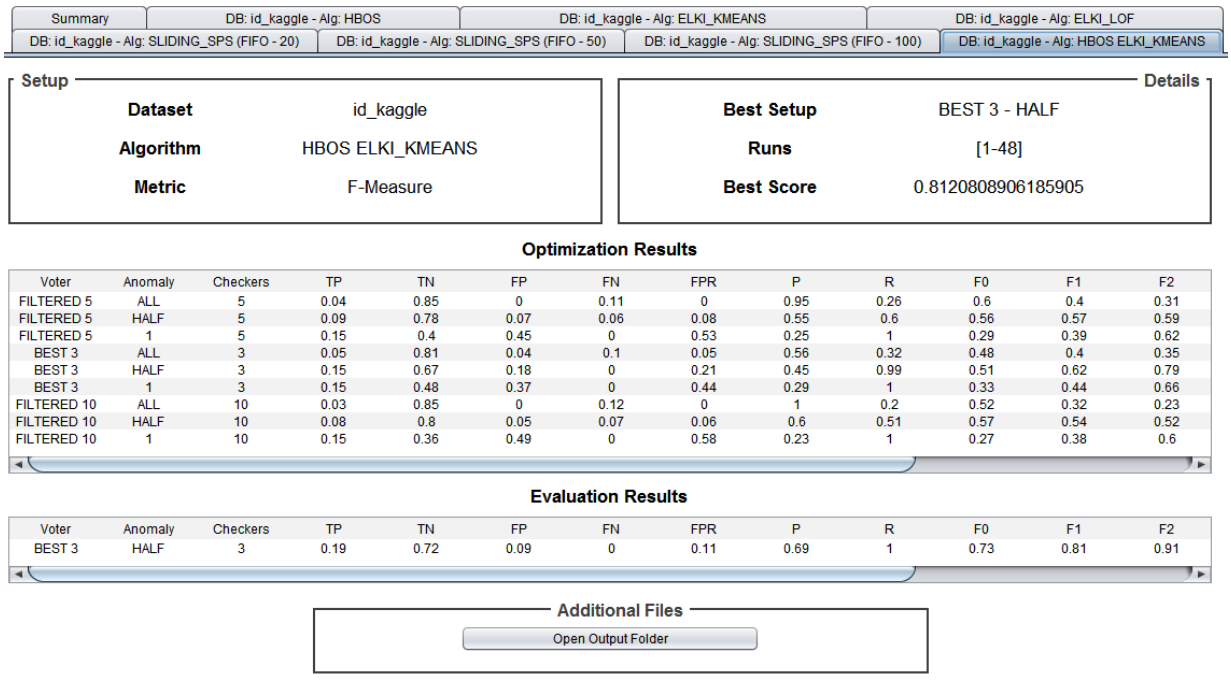
The GUI containing the outputs of the experiments can be seen in Figure 7 and Figure 8.

The first tab to be seen is the "Summary" tab: RELOAD shows a row for each couple <*algorithm*, *data set*> analysed. Each row reports on *i*) the data set, *ii*) the algorithm, *iii*) the most performing pair of selection policy and voting strategy, according to the target metric, *iv*) the percentage of labelled anomalies over all data points



Dataset	Algorithm	Best Configuration	Best Runs	Attacks Ratio	Best Score
id_kaggle	HBOS	BEST 3 - HALF	[1-48]	19.5%	0.72
id_kaggle	ELKI_KMEANS	BEST 3 - HALF	[1-48]	19.5%	0.69
id_kaggle	ELKI_LOF	FILTERED 5 - ALL	[1-48]	19.5%	0.64
id_kaggle	SLIDING_SPS (FIFO - 20)	FILTERED 5 - HALF	[1-48]	19.5%	0.43
id_kaggle	SLIDING_SPS (FIFO - 50)	FILTERED 10 - 1	[1-48]	19.5%	0.38
id_kaggle	SLIDING_SPS (FIFO - 100)	FILTERED 5 - 1	[1-48]	19.5%	0.1
id_kaggle	HBOS ELKI_KMEANS	BEST 3 - HALF	[1-48]	19.5%	0.81

Figure 7. GUI to summarize results of RELOAD.



Optimization Results

Voter	Anomaly	Checkers	TP	TN	FP	FN	FPR	P	R	F0	F1	F2
FILTERED 5	ALL	5	0.04	0.85	0	0.11	0	0.95	0.26	0.6	0.4	0.31
FILTERED 5	HALF	5	0.09	0.78	0.07	0.06	0.08	0.55	0.6	0.56	0.57	0.59
FILTERED 5	1	5	0.15	0.4	0.45	0	0.53	0.25	1	0.29	0.39	0.62
BEST 3	ALL	3	0.05	0.81	0.04	0.1	0.05	0.56	0.32	0.48	0.4	0.35
BEST 3	HALF	3	0.15	0.67	0.18	0	0.21	0.45	0.99	0.51	0.62	0.79
BEST 3	1	3	0.15	0.48	0.37	0	0.44	0.29	1	0.33	0.44	0.66
FILTERED 10	ALL	10	0.03	0.85	0	0.12	0	1	0.2	0.52	0.32	0.23
FILTERED 10	HALF	10	0.08	0.8	0.05	0.07	0.06	0.6	0.51	0.57	0.54	0.52
FILTERED 10	1	10	0.15	0.36	0.49	0	0.58	0.23	1	0.27	0.38	0.6

Evaluation Results

Voter	Anomaly	Checkers	TP	TN	FP	FN	FPR	P	R	F0	F1	F2
BEST 3	HALF	3	0.19	0.72	0.09	0	0.11	0.69	1	0.73	0.81	0.91

Figure 8. GUI to expand on the results of the application of HBOS and KMEANS algorithms to the case study.

in the ED, and *v*) the score of the target metric. More in detail, this view allows to see which algorithm reached higher metric scores, or rather the optimal algorithm for the dataset or the system under investigation. In addition, for each sliding window algorithm the user selected (SPS in the example), RELOAD will show metric scores for each combination of *selection policy* and *window size* the user set through GUI.

As a side note, we can observe how in this case running HBOS and KMEANS algorithm together - combining anomaly checkers using either one of the two algorithms - allowed improving metric scores with respect to executing either HBOS or KMEANS independently. In addition, by looking at SPS results, it is easy to observe that the usage of wider sliding windows does not help SPS in identifying attacks i.e., the wider the window, the lower the F-Measure.

In addition, the GUI shows detail of all combinations of data sets and algorithms for the different selection policies and voting strategies. An example is in Figure 8: given the HBOS and KMEANS algorithm on our case study, it shows the metrics computed for the selection policies BEST 3, FILTERED 5, FILTERED 10 and the voting strategies ALL, HALF, 1. It is worth noticing that the voting strategy severely impacts the performance of the algorithm under evaluation. Intuitively, the 1 strategy reduces the amount of false negatives (FN); instead, achieving consensus among several anomaly checkers composing the ensemble, e.g., the ALL strategy, reduces the number of false alarms (FP). This can also be verified from the FN and FP columns in Figure 8 – see FILTERED 5 – ALL that does not have FPs, while FILTERED 5 – 1 does not have FNs.

References

[1] Chandola, V., Banerjee, A., & Kumar, V. (2009). “Anomaly detection: A survey”. ACM computing surveys (CSUR), 41(3), 15.

[2] Schubert, E., Koos, A., Emrich, T., Züfle, A., Schmid, K. A., & Zimek, A. (2015). A framework for clustering uncertain data. Proceedings of the VLDB Endowment, 8(12), 1976-1979.2010): 10.

[3] Rapid Miner tutorial, https://rapidminer.com/get-started/ [online, last accessed 5th May 2019]

[4] Scikit tutorial, http://scikit-learn.org/stable/documentation.html [online, last accessed 5th May 2019]

[5] Weka, https://www.cs.waikato.ac.nz/ml/index.html [online, last accessed 5th May 2019]

[6] Sokolova M., Japkowicz, S. "Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation" AI 2006 Springer Berlin Heidelberg, 1015-21.

[7] Friedman, Jerome H. "Stochastic gradient boosting." Computational Statistics & Data Analysis 38.4 (2002): 367-378.

[8] Bovenzi, A., Brancati, F., Russo, S., & Bondavalli, A. (2015). An os-level framework for anomaly detection in complex software systems. IEEE Transactions on Dependable and Secure Computing, 12(3), 366-372.

[9] M. Tavallaee, E. Bagheri, W. Lu, and A. A. Ghorbani, “A detailed analysis of the kdd cup 99 data set,” in Computational Intelligence for Securit and Defense Applications, 2009. CISDA 2009. IEEESymposium on. IEEE, 2009, pp. 1–6

[10] A. Shiravi, H. Shiravi, M. Tavallaee, and A. A. Ghorbani, “Toward developing a systematic approach to generate benchmark data sets for intrusion detection,”computers & security, vol. 31, no. 3, pp. 357–374, 2012.

[11] N. Moustafa and J. Slay, “UNSW-NB15: a comprehensive data set for network intrusion detection systems (UNSW-NB15 network data set),” in Military Communications and Information Systems Conference (Mil-CIS), 2015. IEEE, 2015, pp. 1–6.

[12] S. Rosset and A. Inger, “Kdd-cup 99: knowledge discovery in a charitable organization’s donor database.”

[13] G. Creech and J. Hu, “Generation of a new ids test data set: Time to retire the kdd collection,” in Wireless Communications and Networking Conference (WCNC), 2013 IEEE. IEEE, 2013, pp. 4487–4492.

[14] Di Giandomenico, F., and Strigini, L. “Adjudicators for diverse-redundant components”. In Reliable Distributed Systems, 1990. Proceedings., Ninth Symposium on (pp. 114-123). IEEE.

[15] M. Goldstein and S. Uchida. A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data. PloS one, 11(4):e0152173, 2016.

[16] Saeys, Y., Abeel, T., & Van de Peer, Y. (2008, September). Robust feature selection using ensemble feature selection techniques. In Joint European Conference on Machine Learning and Knowledge Discovery in Databases (pp. 313-325). Springer, Berlin, Heidelberg.

[17] GitHub Public Repository *https://github.com/tommyippoz/RELOAD/releases*

[18] L. Zhang, J. Lin, and R. Karim, "Sliding window-based fault detection from high-dimensional data streams", IEEE Transactions on Systems, Man, and Cybernetics, vol. 47, no. 2, pp. 289–303, 2017.

[19] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: identifying density-based local outliers. In ACM sigmod record, volume 29, pages 93–104. ACM, 2000.

[20] M. Goldstein and A. Dengel. Histogram-based outlier score (hbos): A fast unsupervised anomaly detection algorithm. 2012.

[21] J. Tang, Z. Chen, A. W.-C. Fu, and D. W. Cheung. Enhancing effectiveness of outlier detections for low density patterns. In Pacific-Asia Conference on Knowledge Discovery and Data Mining, pages 535–548. Springer, 2002.

[22] S. Ramaswamy, R. Rastogi, and K. Shim. Efficient algorithms for mining outliers from large data sets. In ACM Sigmod Record, volume 29, pages 427–438. ACM, 2000.

[23] M. Amer, M. Goldstein, and S. 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. ACM, 2013.

[24] H.-P. Kriegel, A. Zimek, et al. Angle-based outlier detection in high-dimensional data. In Proceedings of the 14th ACM SIGKDD Int. Conference on Knowledge discovery and data mining, pages 444–452. ACM, 2008.

[25] F. T. Liu, K. M. Ting, and Z.-H. Zhou. Isolation forest. In Data Mining, 2008. ICDM'08. Eighth IEEE Int. Conference on, pages 413–422. IEEE, 2008.

[26] Tang, J., Chen, Z., Fu, A. W. C., & Cheung, D. (2001). A robust outlier detection scheme for large data sets. In In 6th Pacific-Asia Conf. on Knowledge Discovery and Data Mining.

[27] Dupont, Laurent, et al. "Continuous anomaly detection based on behavior modeling and heterogeneous information analysis." U.S. Patent Application No. 12/941,849.

[28] Giotis, Kostas, et al. "Combining OpenFlow and sFlow for an effective and scalable anomaly detection and mitigation mechanism on SDN environments." Computer Networks 62 (2014): 122-136.

[29] Chiappetta, Marco, Erkay Savas, and Cemal Yilmaz. "Real time detection of cache-based side-channel attacks using hardware performance counters." Applied Soft Computing 49 (2016): 1162-1174.

[30] Breiman, Leo. "Random forests." Machine learning 45.1 (2001): 5-32.

[31] A. Lazarevic, L. Ertoz, V. Kumar, A. Ozgur, and J. Srivastava. A comparative study of anomaly detection schemes in network intrusion detection. In Proceedings of the 2003 SIAM Int. Conference on Data Mining, pages 25{36. SIAM, 2003.

[32] Cotroneo, Domenico, Roberto Natella, and Stefano Rosiello. "A fault correlation approach to detect performance anomalies in Virtual Network Function chains." Software Reliability Engineering (ISSRE), 2017 IEEE 28th International Symposium on. IEEE, 2017.

[33] *The paper is currently being published in an international journal. It introduces the concept of anomaly checkers and explores various approaches for voting strategies, albeit for operating at runtime on a target monitor.*

[34] Rodriguez, Juan D., Aritz Perez, and Jose A. Lozano. "Sensitivity analysis of k-fold cross validation in prediction error estimation." IEEE transactions on pattern analysis and machine intelligence 32.3 (2010): 569-575.

[35] Falcão, F., Zoppi, T., Silva, C. B. V., Santos, A., Fonseca, B., Ceccarelli, A., & Bondavalli, A. (2019, April). Quantitative comparison of unsupervised anomaly detection algorithms for intrusion detection. In Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing (pp. 318-327). ACM.

[36] Kaggle – "Intrusion Detection" dataset uploaded by Jinner, https://www.kaggle.com/what0919/intrusion-detection [online, last accessed 5th May 2019]

[37] OpenML portal, https://www.openml.org/ [online, last accessed 5th May 2019]

[38] Zhou, A., Cao, F., Qian, W., & Jin, C. (2008). Tracking clusters in evolving data streams over sliding windows. Knowledge and Information Systems, 15(2), 181-214.

[39] Karimian, S. H., Kelarestaghi, M., & Hashemi, S. (2012, May). I-inclof: improved incremental local outlier detection for data streams. In Artificial Intelligence and Signal Processing (AISP), 2012 16th CSI International Symposium on (pp. 023-028). IEEE.

[40] Zhang, Liangwei, Jing Lin, and Ramin Karim. "Sliding window-based fault detection from high-dimensional data streams." IEEE Transactions on Systems, Man, and Cybernetics: Systems 47.2 (2017): 289-303.

[41] Mouratidis, K., & Papadias, D. (2007). Continuous nearest neighbor queries over sliding windows. IEEE transactions on knowledge and data engineering, 19(6), 789-803.

[42] Ding, Z., & Fei, M. (2013). An anomaly detection approach based on isolation forest algorithm for streaming data using sliding window. IFAC Proceedings Volumes, 46(20), 12-17.