

AutoAD: an Automated Framework for Unsupervised Anomaly Detection

Andrian Putina, Maroua Bahri, Flavia Salutari, Mauro Sozio

▶ **To cite this version:**

Andrian Putina, Maroua Bahri, Flavia Salutari, Mauro Sozio. AutoAD: an Automated Framework for Unsupervised Anomaly Detection. DSAA 2022 - IEEE International Conference on Data Science and Advanced Analytics, Oct 2022, Paris / Virtual Event, France. hal-03811809

HAL Id: hal-03811809

https://inria.hal.science/hal-03811809

Submitted on 12 Oct 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

AutoAD: an Automated Framework for Unsupervised Anomaly Detection

Andrian Putina
LTCI, Télécom Paris
IP-Paris
Palaiseau, France
andrian.putina@telecom-paris.fr

Maroua Bahri
MiMove
Inria Paris
Paris, France
maroua.bahri@inria.fr

Flavia Salutari
LTCI, Télécom Paris
IP-Paris
Palaiseau, France
flavia.salutari@telecom-paris.fr

Mauro Sozio
LTCI, Télécom Paris
IP-Paris
Palaiseau, France
mauro.sozio@telecom-paris.fr

***Abstract*—Over the last decade, we witnessed the proliferation of several machine learning algorithms capable of solving different tasks for the most diverse applications. Often, for an algorithm to be effective, significant human effort is required, in particular for hyper-parameter tuning and data cleaning. Recently, there have been increasing efforts to alleviate such a burden and make machine learning algorithms easier to use for researchers with varying levels of expertise. Nevertheless, the question of whether an efficient and fully generalizable automated Machine Learning (autoML) framework is possible remains unanswered. In this paper, we present autoAD, the first autoML framework for unsupervised anomaly detection. By leveraging a pool of different anomaly detection algorithms, each one coming with its own hyper-parameter search space, our framework automatically selects the best performing approach, while determining an optimal configuration for its hyper-parameters on a given dataset. Our extensive experimental evaluation, conducted on a rich collection of datasets, shows the substantial gains that can be achieved with autoAD compared to state-of-the-art methods for unsupervised anomaly detection.**

***Index Terms*—Anomaly detection, autoML, unsupervised learning.**

I. Introduction

Machine learning is a well established subset of artificial intelligence crawling with strong technical and scientific background. Its success in many application areas yielded to a growing demand for systems that can be used by both experts and novices in the field of machine learning. Anomaly detection[1], also known as outlier detection, is an important data mining and machine learning task that, roughly speaking, consists of identifying data instances that deviate from "normal" behavior. This problem has been widely studied over the last few decades [1], [2] resulting in a large number of algorithms with varying levels of effectiveness. Typical application domains include medical, fraud detection, and intrusion detection. In the literature, there exist a number of studies devoted to unsupervised anomaly detection, which is a relatively common scenario, in that ground truth labels are often unavailable. For instances, iForest [3], OCSVM [4], LOF [5], and more recently, RHF [6].

Often, to unravel the full potential of machine learning, some human expertise and domain knowledge are required in order to select the most effective algorithm, to tune its hyper-parameters, as well as to perform data cleaning (feature selection, dimensionality reduction, etc.). This is often a tedious and non-trivial task for researchers with all levels of expertise. To alleviate such a burden for machine learning users, several techniques have been proposed under the *automated Machine Learning (autoML)* research direction [7], [8]. One of the goals of autoML is to make easier the use of machine learning methods by, among others, automatically selecting the most effective model and its best configuration for its parameters, on given a dataset.

Currently, most efforts in autoML are devoted to supervised machine learning, with very few studies dealing with unsupervised tasks. In particular, no autoML approach for anomaly detection has been proposed, to the best of our knowledge, which is perhaps due to lack of unsupervised evaluation measures [9].

To tackle the aforementioned limitations, we propose autoAD, an automated unsupervised Anomaly Detection framework which selects for a given unlabeled dataset the best performing model, as well as its optimal configuration for its parameters. Motivated by the established supervised autoML frameworks, *e.g.*, , Auto-WEKA [10] and Auto-Sklearn [11], the proposed autoAD framework involves different outlier detection algorithms and their corresponding hyper-parameter search spaces. Given an input dataset, autoAD applies the algorithms with their different hyper-parameter configurations in parallel, afterwards it evaluates the performance using an unsupervised evaluation strategy that we propose which is based on anomalies removal.

The main contributions of this paper are summarized as follows:

- We develop autoAD, a framework for automated unsupervised anomaly detection, which, to the best of our knowledge, represents the first autoML approach for unsupervised anomaly detection.
- We propose an unsupervised metric strategy that permits the evaluation of anomaly detectors in a fully unsupervised way by removing outliers and using statistical measures, such as kurtosis and variance, on the remaining normal instances.

[1]In compliance with literature terminology, we use the terms *anomaly* and *outlier* interchangeably.

- We conduct an extensive experimental evaluation on a diverse set of datasets which shows that `autoAD` achieves significantly better performances than using a single anomaly detection algorithm. We also show the similarity in results of our proposed unsupervised evaluation strategy in comparison to what would be obtained with commonly used metrics to evaluate anomaly detectors, such as area under curve.
- To foster reproducibility, the code and datasets employed in our work are available at http://bit.ly/3mdT6qu.

The reminder of this work is organized as follows. In Section II, we detail related work where we present well-known anomaly detection algorithms followed by the state-of-the-art in AutoML. Section III presents the description of our proposed framework and the employed methodology for unsupervised anomaly detection evaluation. In Section IV, we show the experimental results and discussions. We finally end this paper by drawing conclusions and future directions in Section V.

II. Related Work

A. Anomaly Detection

Anomaly detection is one of the most widely studied problem in machine learning. It consists of finding the instances which substantially deviate from the other observations, named anomalies or outliers. The identification of the anomalies is essential in diverse application domains, ranging from data security and fraud detection to healthcare. Due to the lack of available labeled datasets, unsupervised anomaly detection overtook its supervised counterpart, receiving increasing attention in recent years by the research community.

Anomaly detection algorithms can be essentially grouped in three classes: *Proximity/Nearest Neighbor*-based methods (*e.g.*, k-NN, LOF, *etc.*), *Probabilistic/Linear*-based methods (*e.g.*, PPCA, OCSVM, *etc.*), and *Ensemble/Isolation*-based methods. The latter category includes algorithms that, instead of profiling normal instances, isolate the anomalies by means of recursive splitting over the data through a random tree and by generating isolation forests.

Among them, it is worth mentioning isolation Forest (iForest) [3] which has consistently proven to be one of the most effective algorithms for unsupervised anomaly detection [6], [12], [13]. Given a dataset, iForest builds a forest of randomly generated trees and assigns to each instance x the anomaly score, which is the average path length from the root to the node containing x. The input parameters of iForest are the sub-sampling size ψ, and the number of trees. In [3], authors show empirically that the anomalies are those instances having shorter path lengths, as they are more likely to be isolated.

Random Histogram Forest (RHF) [6] is another ensemble method which is based on a forest of *weak* trees. RHF builds each tree by recursively splitting the dataset according to the kurtosis of the attributes and on a split value randomly selected. Finally, it assigns the anomaly score to each instance according to the information-content of the leaf it belongs to, aggregated over all the ensemble trees. RHF depends on the number of trees and on the maximum height h of the trees which indicates the maximum number of 2^h leafs each tree produces, *i.e.*, the bins in which all the instances are grouped.

B. Automated Machine Learning

The performance of a given machine learning method depends on the quality of the algorithm as well as its parameterization, a task which is sometimes difficult to fix to the optimal values. AutoML [7], [8] is a new topic that supports researchers and practitioners with the tedious work of manually designing machine learning pipelines, which include performing algorithm selection and tuning hyper-parameters. AutoML can be also viewed as the process that makes machine learning easier by avoiding manual hyper-parameters tuning for both machine learning experts and non-experts.

This very hot topic has attracted several researchers during the recent years due to the importance of its application. In fact, the basic autoML algorithms have been initially proposed for the supervised learning and are discussed in recent surveys on autoML and its open challenges [14], [15], [16]. Examples of well-known autoML approaches are (i) *irace* [17] that uses an iterated racing procedure where the worst configurations are replaced by new ones for each iteration (race); (ii) *SMAC* [18] that performs a Bayesian optimization in conjunction with a simple racing mechanism on the instances to efficiently decide which of two configurations performs better; and (iii) *ParamILS* [19] which is based on an iterated local search that starts by evaluating the default and some other configurations on a subset of instances, then the best configuration will be maintained and tested on a different subset of data.

Throughout the last five years, multiple tools and systems have been developed which serve autoML [16], [20], [21]. For instance, `Auto-Sklearn`[2] [11] and `Auto-WEKA`[3] [10] which are two automated systems that have been implemented on top of the well-known machine learning softwares `scikit-learn` and `WEKA`, respectively. These tools and techniques exclusively deal with supervised methods where ground truth labels are used during the model selection and the hyper-parameters tuning processes.

Nevertheless, anomaly detection algorithms tend to be highly sensitive to their hyper-parameterization that might potentially affect the final results. In order to improve the latter, previous works have investigated the use of meta-learning[4] [22], [9] for the anomaly detection task [23], [24]. The algorithms used for the meta-learning might be unsupervised, but the meta-learning evaluation is supervised, *i.e.*, the evaluation measures used to evaluate the performance of meta-learners on the meta-data require access to labels. The most commonly used metric for anomaly detection evaluation is the *Area Under the Curve (AUC)* of the *Receiver Operating*

[2]https://www.automl.org/automl/auto-sklearn/

[3]http://www.cs.ubc.ca/labs/beta/Projects/autoweka/

[4]Meta-learning starts by applying algorithms to meta-data, then, given a test dataset, searches for the most similar dataset in the meta-data and its corresponding best performing algorithm.

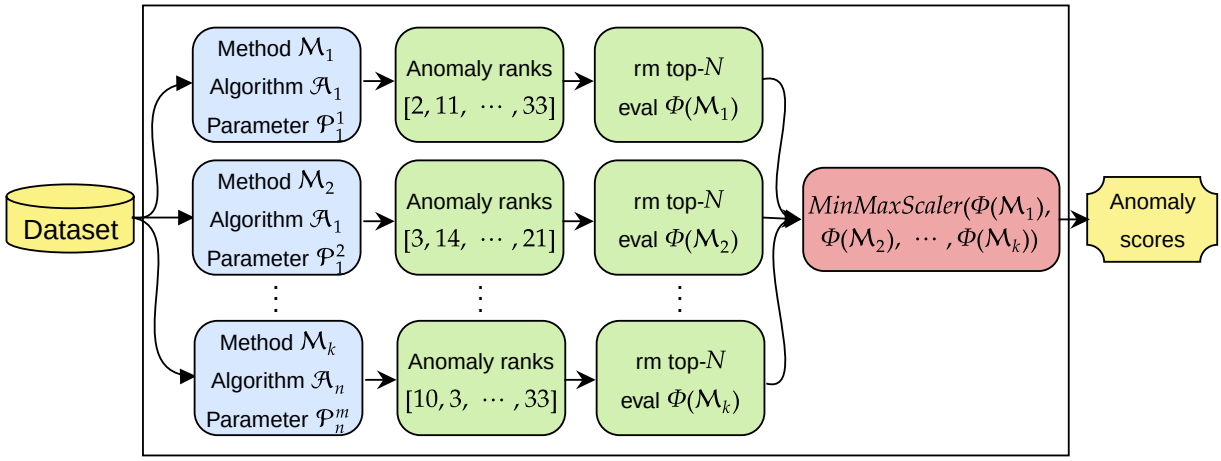


Fig. 1: Overview of the autoAD framework. Given the input dataset and the set of algorithms with their corresponding predefined search space, we apply separately each method, as the combination $(\mathcal{A}_i, \mathcal{P}_i^j)$. Once we finish processing each of the method, we rank the output anomaly scores (as shown in the first green column cells) in order to remove (*rm*) the top-N anomalous instances. We therefore evaluate (*eval*) the performance of each method using a quality measure, ϕ, on the remaining normal instances. We assign to each method a weight proportional to its performance. Hence, the final anomaly scores are computed based on the initial scores and the weight assigned to each method.

Characteristic (ROC) [6], [25]. However, when data labels are imbalanced, measuring the AUC under the precision recall curve, known as *Average Precision (AP)*, can give better insight about the algorithm performance [26]. On the other hand, clustering quality metrics cannot be directly used to evaluate anomaly detection algorithms because the latter are scoring and ranking the instances while clustering algorithms are grouping them. In this work, we aim to propose an effective way to measure the quality of unsupervised anomaly detection algorithms without knowledge of ground truth labels.

III. The Proposed autoAD Framework

In this section, we present an automated unsupervised anomaly detection framework that aims to find the optimal anomaly detection algorithm and its best configuration for a given unlabeled dataset.

An effective automated framework for unsupervised anomaly detection should be composed of a set of different unsupervised detectors with distinct configurations. The proposed autoAD framework consists in four key steps: (i) application of the anomaly detection algorithms where each one outputs the anomaly scores of instances given a dataset; (ii) removing the top N anomalous instances, *i.e.*, instances with the highest anomaly scores; (iii) assigning a weight to each algorithm using a given evaluation metric; and (iv) obtaining the final anomaly scores for each instance. An overview of autoAD is given in Fig. 1.

A. Application of Anomaly Detection

Each outlier detection method in our automated framework is composed of the algorithm and its hyper-parameter configuration. Let $(\mathcal{A}, \mathcal{P})$ be the method $\mathcal{M}$ that uses algorithm $\mathcal{A}$ with hyper-parameters $\mathcal{P}$. We define $\mathbb{M}$ the set of methods which is composed of $k = |\mathbb{M}|$ combinations of each algorithm tuned with each of its corresponding hyper-parameter as follows:

$$\mathbb{M} = \mathcal{M}_1, \mathcal{M}_2, \cdots, \mathcal{M}_k = (\mathcal{A}_1, \mathcal{P}_1^1), (\mathcal{A}_1, \mathcal{P}_1^2), ..., (\mathcal{A}_n, \mathcal{P}_n^m), \quad \forall \mathcal{A}_i \in \mathbb{A}, \mathcal{P}_i^j \in \mathbb{P},$$

where $\mathbb{A}$ is a set of anomaly detection algorithms and $\mathbb{P}$ is a set of predefined hyper-parameters for the algorithms. Each algorithm $\mathcal{A}_i$ is used with a different hyper-parameter configuration j in our framework.

Given an input unlabeled dataset, each method in the set $\mathbb{M}$, $(\mathcal{A}_i, \mathcal{P}_i^j)$, is simultaneously used to build a model and accordingly compute the output anomaly score of each instance. Generally, at this stage, common state-of-the-art ensemble methods and frameworks use a metric to evaluate the quality of the results. However, the most widely used metric so far is the AUC ROC/AP which uses labels to assess the performance. This technique is commonly used in the literature to evaluate both supervised and unsupervised anomaly detection algorithms [6], [25].

In the following steps, we gradually present our key strategy in evaluating unsupervised detectors in a fully unsupervised way that mostly corresponds to performance in terms of AUC/AP criteria (see empirical studies in Section IV). In this framework, we used two main algorithms, Random Histogram Forest [6] and Isolation Forest [3], $\mathbb{A} = \{RHF, iForest\}$, because they proved to be the best performing ones according to a recent empirical study [6]. Of course, more algorithms can be easily added with their hyper-parameters' space to our autoAD framework.

B. Anomalies Removal

Anomaly algorithms assign a score to each instance in a given dataset that can be used to rank instances depending on their level of outlierness, *i.e.*, the anomalies are ranked before the non-anomalous instances (the most anomalous instance has the biggest score and is ranked first). Algorithm 1 presents the pseudocode of how we remove anomalies. In fact, the score result obtained from each method in `autoAD` will be ranked from the highest to the smallest one (line 1, Algorithm 1). In this work, we propose a quality measure that does not require data with known labels. This evaluation strategy starts by removing the top N anomalous instances and evaluates the remaining "normal" instances using a quality metric. Some questions naturally arise: *How many instances of the top ranked ones are actually anomalies? and how can we fix N?* This can be tricky as it involves a parameter that controls the number of anomalies. In the unsupervised context, we assume that the right percentage of anomalies in a given dataset is unknown. An envisaged solution to answer these questions consists in picking a random value

$$N = \mathcal{U}(0, 10\%),$$

and removing the N top ranked instances, where N is between $[0, 10\%]$.

To avoid picking an unreasonable value, this process is repeated $r = 100$ times (line 3, Algorithm 1), *i.e.*, we do 100 runs and randomly select N for each time. In the end, we average the results of the different runs to appropriately capture the anomalies.

Once we remove the N-top ranked anomalies (line 4, Algorithm 1), we separately compute the performance of all the methods (line 5, Algorithm 1) using a quality measure ϕ, such as kurtosis, variance, or Error Sum of Squares (SSE), on the remaining – *supposed to be* – normal non-anomalous instances, as depicted in Fig. 2. The final value of ϕ, for each method in our `autoAD`, is used to normalize the instances' score obtained in Section III-A.

Algorithm 1: AnomaliesRemoval(X, ms_i, R)

Input: X: dataset ; ms_i: anomaly scores of method i; R: list of N-top ranked instances to remove drawn from $\mathcal{U}(0, 10\%)$

Output: Quality measure ϕ_i

1: Ranks = SortRank(ms_i)// sort instances according to their anomaly scores in descending order
2: $\phi_i = 0$
3: **for** all $N \in R$ **do**
4: $X_{filtered}$ = filter(X $\setminus Ranks$[1:N]) // remove the top N most anomalous instances from X
5: ϕ_i += QualityMeasure($X_{filtered}$) // compute the quality measure on the filtered dataset and aggregate over R
6: **end for**
7: return ϕ_i

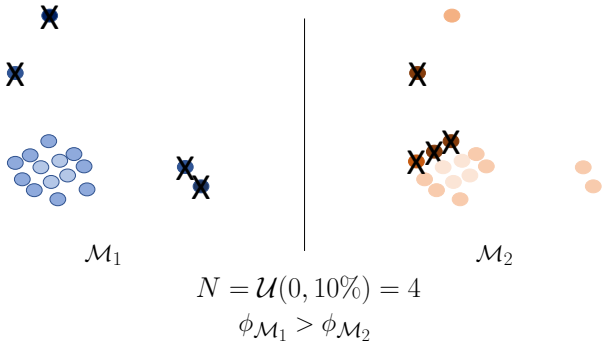


Fig. 2: *Anomalies Removal* example applied on two different methods $\mathcal{M}_1$ and $\mathcal{M}_2$. The two methods produce different anomaly scores and ranks as illustrated by the color of each instance (darker colors indicate higher anomaly scores). By removing, for example, $N = 4$ most anomalous instances and applying a quality metric on the remaining instances $\mathcal{M}_1$ proves to be better than $\mathcal{M}_2$.

Quality metrics. Several quality metrics can be defined and used in the `autoAD` framework. As there are many types of anomalies (*e.g.*, *global*, *local* or *contextual*), it is possible to design quality metrics whose goal is to target a particular kind of anomalies. In this work we focus mainly on *global* anomalies that usually lie in the tails of the data distribution. Starting from these characteristics, we define three quality measures that serve the automated method to understand which method better compresses the data after removing the N-top ranked instances:

- *SSE*: The simplest measurement one can compute on the – supposed to be – normal instances is the SSE. The method whose SSE measurement is the smallest have to be considered the best as it better compresses the data after removing the anomalous instances. By computing such a measure, we assume that normal instances are drawn from a unique cluster (not always true). Moreover, SSE can be weak when dealing with high dimensional datasets.
- *VAR*: The variance is one of the most used dispersion metrics. Unlike the previous measure, the variance is dimension-wise and can be obtained by computing the variance of each dimension and aggregating the scores. Similarly to the previous case, the method whose aggregated variance is the smallest, after removing the N-top ranked anomalies, has to be considered the best method.
- *KURT*: The kurtosis measures the tailedness of the data and can be interpreted as a metric of outliers in the tail. Similarly to the *VAR*, we measure the kurtosis of each dimension after removing the N-top ranked anomalies and aggregate the scores. The method whose aggregated kurtosis is smaller after removing the N-top ranked

anomalies has to be considered the best method.

C. Algorithm Weighting

Each method in $\mathbb{M}$ produces its own measure ϕ obtaining so:

$$\Phi = \{\phi_1,\ \phi_2,\ \phi_3,\ ...\ \phi_k\},$$

where $k = |\mathbb{M}|$. The best method in the set is the one obtaining the best quality measure (*e.g.*, the method whose variance is the smallest after removing the N-top ranked instances) while the worst one is the one obtaining the worst quality measure.

A weight proportional to the measure ϕ_i is assigned to each method $\mathcal{M}_i$ by normalizing (min/max) the set of measures Φ.

$$W = MinMaxScaler(\Phi) = \{w_1,\ w_2\ w_3\ ...\ w_k\}.$$

The best quality measure (*e.g.*, the minimum kurtosis) originates the best weight $w_i = 1$, the worst one $w_i = 0$ while the remaining ones $w_i \in [0, 1]$.

D. Final Anomaly Scores

The final anomaly scores take into account the initial anomaly scores of each method in $\mathbb{M}$ and the weights assigned by the quality measure in the previous step. As the output anomaly scores of each algorithm can be homogeneous and represented in very different ranges we scale all of them between 0 and 1. Subsequently, the anomaly scores of each method undergo the weighing process in which the weights produced in the quality measure phase are used. The final anomaly score is obtained aggregating the results over all the methods as follows:

$$S = \sum_{i=1}^{|\mathbb{M}|} ms_i w_i,$$

where ms_i corresponds to the initial anomaly score of method i while w_i corresponds to its weight.

IV. Experimental Evaluation

We conduct an extensive experimental evaluation to assess the effectiveness of our approach. All algorithms developed in our work are publicly available, so as to foster reproducibility [5]. We implemented our algorithms in Python 3.8 and used iForest available in the scikit-learn library [27] and RHF[6].

A. Datasets

We consider a diverse set of datasets coming from different data sources, with different size and anomaly ratio. Most of them have been widely used as benchmarks when evaluating anomaly detection algorithms. In particular, we consider 16 datasets that are publicly available the UCI [28] or the ODDS [29] repositories. The size of our datasets range from 351 to 623091 instances (n), while the number of dimensions vary from 3 to 274(d). The anomaly ratio is between 0.03% up to 35.9%. Similarly to [6], we use well-known datasets such as *ionosphere, arrhytmia, satellite, mnist, shuttle, mulcross* and some extracted from the *KDD99* dataset. Table I presents the statics of our datasets.

The KDD'99 Cup dataset is one of the most widely used benchmark for anomaly detection. Similarly to the filtering technique used by [30], [31] we extract 5 subsets according to the values of the *service* attribute (*http*, *smtp*, *ftp*, *finger* and *other*). Out of the 41 available attributes, we select, similar to [31], only 3 of them namely "duration", "source bytes", and "destination bytes" as they are thought to be the most relevant ones [31]. We obtain in this way the datasets we call *kdd_http*, *kdd_smtp*, *kdd_ftp*, *kdd_finger*, and *kdd_other*. While [30] filtered the dataset according to the *service* attribute only, [31] filters them also by the positive *logged_in* attribute as they are successful attacks. We also consider this additional filter by further reducing the *kdd_http* dataset into the *http_logged* one by excluding the negative values of *logged_in* attribute.

In order to determine to which extent the presence of duplicates might affect the overall results, we consider also a smaller version in which duplicates have been filtered out: we will refer to them as *kdd_http_distinct*, *kdd_smtp_distinct* and *kdd_ftp_distinct*. We include in our comparison also the full version *kdd_http29* and *kdd_smtp29* in which all the 29 continuous attributes are used. All the continuous features are used also by [30] in which the authors tackle also the duplicates problem by limiting the number of attacks and present to the community their kdd99 dataset (composed by 620098 instances with 0.17% *anomalous* instances). We will refer to this dataset as *kdd99* by author's name.

All such datasets are widely used as benchmarks in the evaluation of anomaly detection methods [6], [3], however, our work is the first that considers all of them in a same experimental evaluation. This is important to provide a fair comparison of the different methods, while highlighting their strengths and weaknesses.

B. Methods and parameters

In our approach `autoAD`, we use both RHF and iForest (denoted ISO in the following) algorithms as the main anomaly detection engines. For each each of them we consider 8 different values for their main parameter, namely, *sampling size* ψ for ISO and *maximum height* h for RHF. In both cases, we employ the same number of trees $t = 100$. Based on parameters range, authors' insights and considerations done in the original paper of the two algorithms, we select $h \in [1, 8]$ for RHF and $\psi \in [32, 64, 128, ..., 4096]$ for iForest.

We compare `autoAD` against the ensemble $\{RHF + ISO\}_{dft}$ in which the two methods are weighted equally, while using their default parameters, namely, $h = 5$ and $\psi = 256$, respectively. We also compare our approach against the ensemble $\{RHF+ISO\}_{all}$ (with same weight) consisting of all possible 16 parameter configurations for RHF and ISO. Furthermore, we report the results produced by RHF and ISO, while using their default parameters.

[5]The source code and datasets employed in our analysis can be found at http://bit.ly/3mdT6qu.

[6]https://github.com/anrputina/rhf

dataset	n	d	anomalies(%)	dataset	n	d	anomalies(%)
ionosphere	351	33	126 (35.9%)	shuttle	49097	9	3511 (7.15%)
wbc	378	30	21 (5.56%)	smtp29	96554	29	1183 (1.23%)
arrhytmia	452	274	66 (14.6%)	http_distinct	222027	3	75 (0.03%)
cardio	1831	21	176 (9.61%)	mulcross	262144	4	26214 (10.0%)
musk	3062	166	97 (3.17%)	cover	286048	10	2747 (0.96%)
satellite	5100	36	75 (1.47%)	http_logged	567498	3	2211 (0.39%)
satimages	5803	36	71 (1.22%)	kdd99	620098	29	1052 (0.17%)
mnist	7603	100	700 (9.21%)	http29	623091	29	4045 (0.65%)

TABLE I: Overview of the datasets. For each dataset, we have the number of instances n, number of dimensions d, and number of anomalies (in $\%$).

Method	Median Gain	$\{RHF + ISO\}_{dft}$			
		Win Rate	Med Gain	Loss Rate	Med Loss
autoAD-SSE	+21%	37.5%	+35%	37.5%	-1.4%
autoAD-VAR	+21%	43.5%	+30%	37.5%	-1.5%
autoAD-KURT	+22%	50%	+26%	31%	-1.6%

TABLE II: The overall median gain of autoAD using the three different quality measures with the respect to $\{RHF + ISO\}_{dft}$. The Win/Loss rate indicates the number of datasets in which autoAD is statistically better/worse. The Med Gain/Loss indicates the median AP gain in the winning/worsening datasets.

C. Results

We report the average results over 30 runs for each of the randomized algorithms: RHF, ISO, and the *AnomaliesRemoval* function (Algorithm 1). The statistics are generated using the two sample Kolmogorov-Smirnov test ($\alpha = 0.05$) to check if the output of the two models are drawn from the same distribution while the Welch's two-tailed t-test is used to check if they have the same mean. Table II shows a summary of the results achieved by autoAD using the three quality metrics SSE/VAR/KURT versus $\{RHF + ISO\}_{dft}$. Table III presents the detailed results in terms of average precision (AP) and its confidence interval computed with a 95% confidence level obtained by each model on the different datasets.

We observe that for all three quality measures, we obtain median gain of 21% (SSE and VAR) up to 22% (KURT) in terms of AP while being winners on 37.5% (SSE) up to 50% (KURT) of the datasets. On the other hand, we see that $\{RHF + ISO\}_{dft}$ can perform better than our proposed method in at most 37.5% of the datasets while the two methods produce the same results on 20% of the datasets (not shown in the table). We notice that, the median AP increases in the winning datasets by at least 26% (KURT) up to 35% (SSE). autoAD-KURT seems to be the best metric as it produces the best results on 50% of the datasets with a median gain of 22%.

For example, in the kdd99 and mulcross datasets, the $\{RHF + ISO\}_{dft}$ ensemble reaches an AP score of 0.566 ± 0.005 and 0.657 ± 0.019 while autoAD-KURT scores 0.798 ± 0.011 and 0.810 ± 0.008 achieving a gain of 41% and 23% respectively (see Table III). On the other hand, when $\{RHF + ISO\}_{dft}$ is better than autoAD-KURT (31% of the datasets), the median loss is very limited and has only a limited impact on the final result. For instance, with smtp29 (0.984 ± 0.001 vs. 0.974 ± 0.001), satimages (0.929 ± 0.003 vs. 0.922 ± 0.001), shuttle (0.968 ± 0.001 vs. 0.952 ± 0.001) or ionosphere (0.810 ± 0.002 vs. 0.807 ± 0.002) the losses are -1%, -0.7%, -1.6% and -0.4% respectively.

To understand such important improvement, we study the impact of our main parameters, the number of removals r and the number of methods k, on the precision performance. Fig. 3 shows how the AP score fluctuates when changing r on three datasets. We notice that when r increases (where we remove the instances with the highest anomaly scores), the AP score increases accordingly. That is an expected behavior similar to ensemble-based methods where several weak learners are more accurate than a single one. Although we observe for most of the datasets steady results already for $30 < r < 50$, thus we set $r = 100$ as it is commonly used in ensemble models.

In Fig. 4, we present the autoAD results in terms of precision while increasing the number of methods used in our framework. We remark that the overall precision tends to be as good as the best method in the ensemble even when only few methods are used. For instance, autoAD performs as good as *ISO@256*[7] (best method) when the first four methods are used. By further growing the number of methods, the space of choices broadens and the model starts showing results better than any single method.

Quality measures. The overall unsupervised quality metric ϕ plays a key role in the weighting phase. A good correlation between the latter and the average precision is therefore desirable for the autoAD to produce satisfying results.

We report in Fig. 5 two examples on kdd99 and mulcross datasets showing the assigned weight w, the AP score for each method used and its quality measure ϕ. The methods are ordered according to the quality measure in decreasing order. Taking a deeper look at Fig. 5 (a) and (b) with the kdd99

[7]Algorithm@parameter. Example: $ISO@256$ is the iForest algorithm with $\psi = 256$.

dataset	autoAD-SSE	autoAD-VAR	autoAD-KURT	$\{RHF + ISO\}_{dft}$	$\{RHF + ISO\}_{all}$	RHF	ISO
ionosphere	0.804 ± 0.001	0.806 ± 0.001	0.807 ± 0.001	$\mathbf{0.810 \pm 0.002}$	0.803 ± 0.001	0.810 ± 0.004	0.808 ± 0.002
wbc	0.585 ± 0.005	0.587 ± 0.005	0.583 ± 0.005	0.586 ± 0.009	0.593 ± 0.005	0.567 ± 0.014	0.598 ± 0.010
arrhytmia	0.457 ± 0.003	0.456 ± 0.003	$\mathbf{0.458 \pm 0.003}$	0.454 ± 0.007	0.466 ± 0.003	0.426 ± 0.010	0.463 ± 0.007
cardio	0.587 ± 0.005	0.586 ± 0.005	0.582 ± 0.004	0.582 ± 0.009	0.590 ± 0.004	0.582 ± 0.014	0.551 ± 0.015
musk	1.000 ± 0.000	1.000 ± 0.000	$\mathbf{1.000 \pm 0.000}$	0.999 ± 0.001	1.000 ± 0.000	0.997 ± 0.004	0.995 ± 0.003
satellite	0.643 ± 0.003	0.642 ± 0.004	0.648 ± 0.003	0.649 ± 0.005	0.654 ± 0.002	0.644 ± 0.008	0.641 ± 0.006
satimages	0.918 ± 0.002	0.918 ± 0.002	0.922 ± 0.001	$\mathbf{0.929 \pm 0.003}$	0.931 ± 0.001	0.926 ± 0.004	0.921 ± 0.004
mnist	0.365 ± 0.006	0.366 ± 0.006	$\mathbf{0.366 \pm 0.006}$	0.333 ± 0.012	0.335 ± 0.004	0.363 ± 0.018	0.272 ± 0.010
shuttle	0.952 ± 0.002	0.950 ± 0.001	0.952 ± 0.001	$\mathbf{0.968 \pm 0.001}$	0.966 ± 0.001	0.935 ± 0.002	0.978 ± 0.002
smtp29	0.976 ± 0.001	0.974 ± 0.001	0.974 ± 0.001	$\mathbf{0.984 \pm 0.001}$	0.982 ± 0.001	0.946 ± 0.011	0.989 ± 0.001
http_distinct	0.605 ± 0.010	0.692 ± 0.007	$\mathbf{0.694 \pm 0.007}$	0.618 ± 0.018	0.649 ± 0.008	0.761 ± 0.009	0.021 ± 0.003
mulcross	0.855 ± 0.009	0.857 ± 0.009	$\mathbf{0.81 \pm 0.008}$	0.657 ± 0.019	0.702 ± 0.009	0.753 ± 0.015	0.551 ± 0.028
cover	0.059 ± 0.002	0.061 ± 0.002	$\mathbf{0.066 \pm 0.003}$	0.061 ± 0.005	0.058 ± 0.002	0.077 ± 0.010	0.057 ± 0.007
http_logged	0.962 ± 0.002	0.966 ± 0.001	0.967 ± 0.001	$\mathbf{0.973 \pm 0.002}$	0.968 ± 0.001	0.966 ± 0.002	0.907 ± 0.040
kdd99	0.794 ± 0.011	0.791 ± 0.011	$\mathbf{0.798 \pm 0.011}$	0.566 ± 0.005	0.593 ± 0.005	0.787 ± 0.023	0.531 ± 0.002
http29	0.750 ± 0.016	0.750 ± 0.017	$\mathbf{0.753 \pm 0.018}$	0.577 ± 0.017	0.584 ± 0.008	0.699 ± 0.044	0.546 ± 0.016
average	0.70	0.71	**0.71**	0.67	0.67	0.70	0.61
median	0.77	0.77	**0.77**	0.63	0.65	0.75	0.58

TABLE III: Average precision scores of all approaches on all the datasets. autoAD-SSE, autoAD-VAR and autoAD-KURT represent the autoAD algorithm using the three different quality measures. $\{RHF + ISO\}_{dft}$ is the ensemble composed by *RHF* and *iForest* using their default parameters while $\{RHF + ISO\}_{all}$ is the ensemble composed by all the 16 methods (with the different configurations) used in autoAD without the outlier removal strategy. *RHF* and *ISO* are the two methods using their default parameter. **Bold** values are the best between autoAD-KURT and $\{RHF + ISO\}_{dft}$.

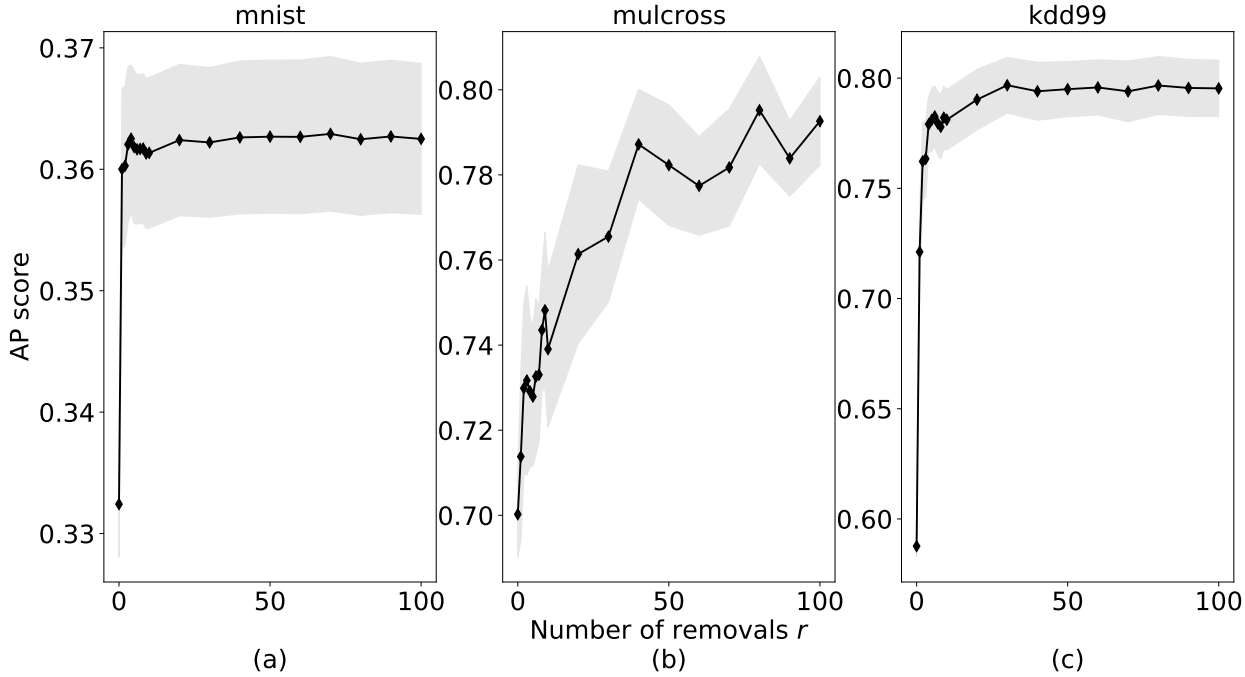


Fig. 3: Average precision score of autoAD for an increasing number of removals r on three different datasets, (a) mnist, (b) mulcross, and (c) kdd99 datasets, respectively.

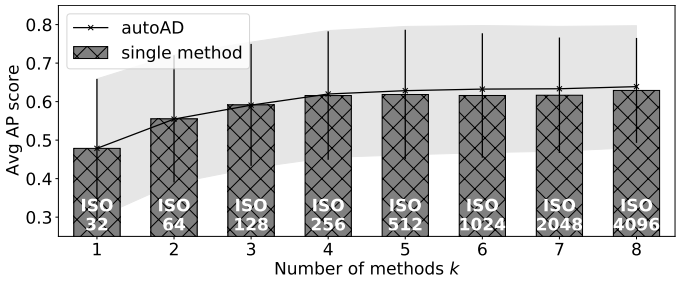


Fig. 4: Average precision scores with a different number of methods. By using a higher number of methods the model starts showing better results than any single method.

dataset, one can notice that the worst performing method, *i.e.*, *ISO@32* with an AP=0.17, is also the one that produces the worst result in terms of the quality measure ϕ, where the weight $w = 0$. By looking at the quality measure plot from left to right, we observe that decreasing values (and thus higher weight w) correlate with increasing AP scores. Similar result is obtained with the mulcross dataset represented in Fig. 5 (c) and (d), as well as with most of the datasets. Similarly, in this case larger weights are assigned to methods having high AP scores.

Interestingly, we remark a contradictory pattern when comparing the weights assigned to each method. For example, with the kdd99 dataset, we obtain better precision when we increase the sampling parameter ψ. The worst result is thus achieved by *ISO@32* while *ISO@4096* gives the best performance. Opposite results were obtained with the mulcross dataset, where *ISO@4096* is the worst and *ISO@32* is one of the best performing methods. Similarly, while *RHF@1* (AP = 0.7) seems to be one of the worst parameters in the first case and *RHF@8* (AP=0.8) the best one, the opposite is true in the mulcross dataset where *RHF@8* achieves a score of only AP=0.5 in contrast to AP=0.9 of *RHF@1*.

Running Time. The running time of autoAD also depends on the number of input methods $k = |\mathbb{M}|$ and the number of removals drawn at random from $\mathcal{U}(0, 10\%)$.

In Fig. 6, we report the running time behavior when we change these two parameters. Fig. 6 (a) depicts the running time consumed by autoAD and each method in the framework. We can see that the overall running time of autoAD increases linearly with time performance of each method in the ensemble. On the other hand, Fig. 6 (b) shows that when we increase the number of removals r, the execution time of autoAD linearly increases accordingly.

D. Discussions

The autoAD framework shows a high correlation between the obtained AP scores and the three quality measures we propose producing up to 41% of overall gain on some datasets, while keeping the loss very limited when failing (around -1%). The results obtained using three different unsupervised quality metrics show that it is sufficient to focus on the top most anomalous scores of each method to study its reliability and effectiveness.

V. Concluding Remarks and Future Directions

In this paper, we presented autoAD, the first, to the best of our knowledge, autoML framework for unsupervised anomaly detection based on anomalies removal which produces simple

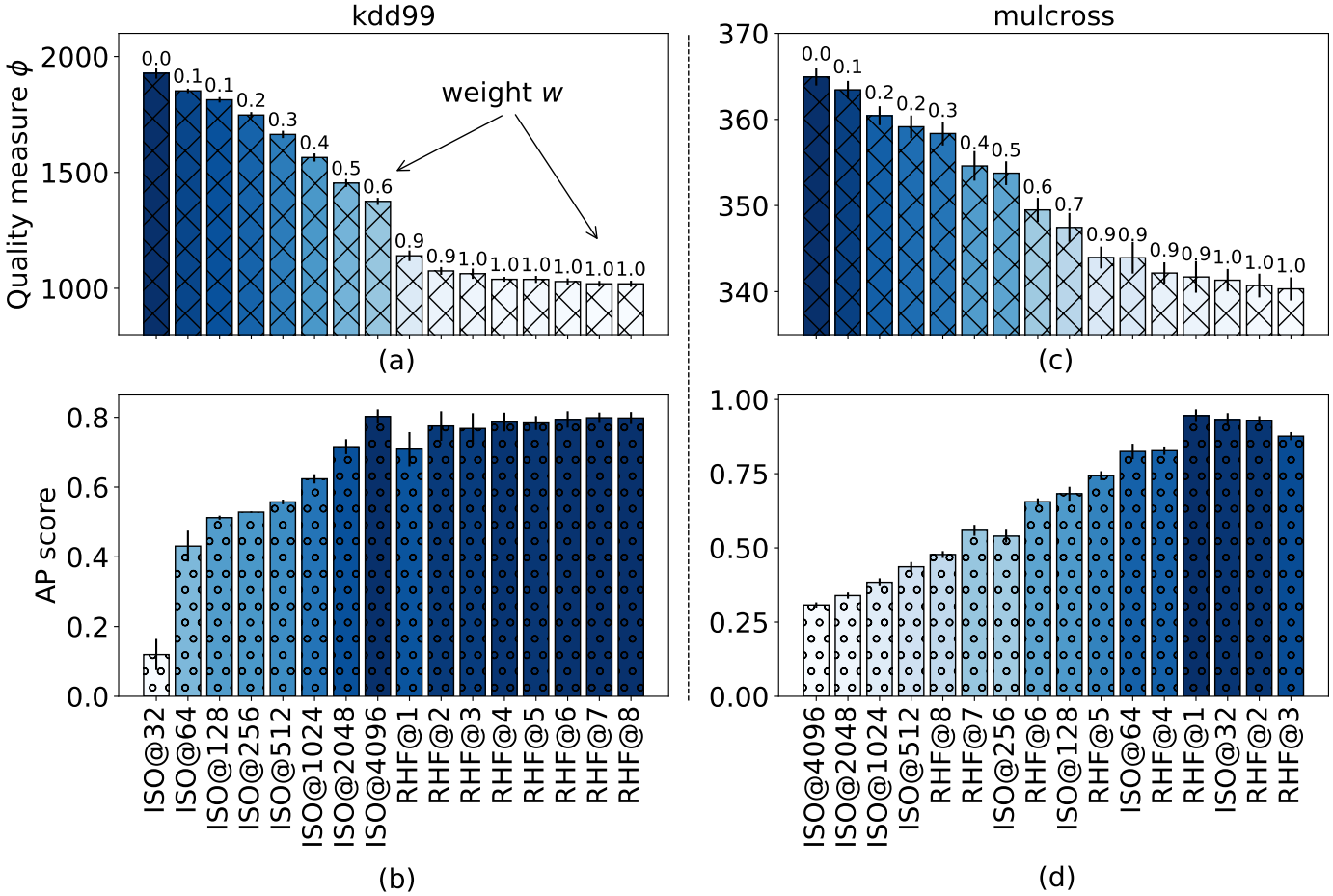


Fig. 5: Correlation between the kurtosis-quality measure ϕ with (a) kdd99, (c) mulcross, and the average precision score with (b) kdd99, (d) mulcross of each method. The methods are ordered according to their quality measure in a decreasing order from the highest ϕ (worst value) to the smallest ϕ (best value).

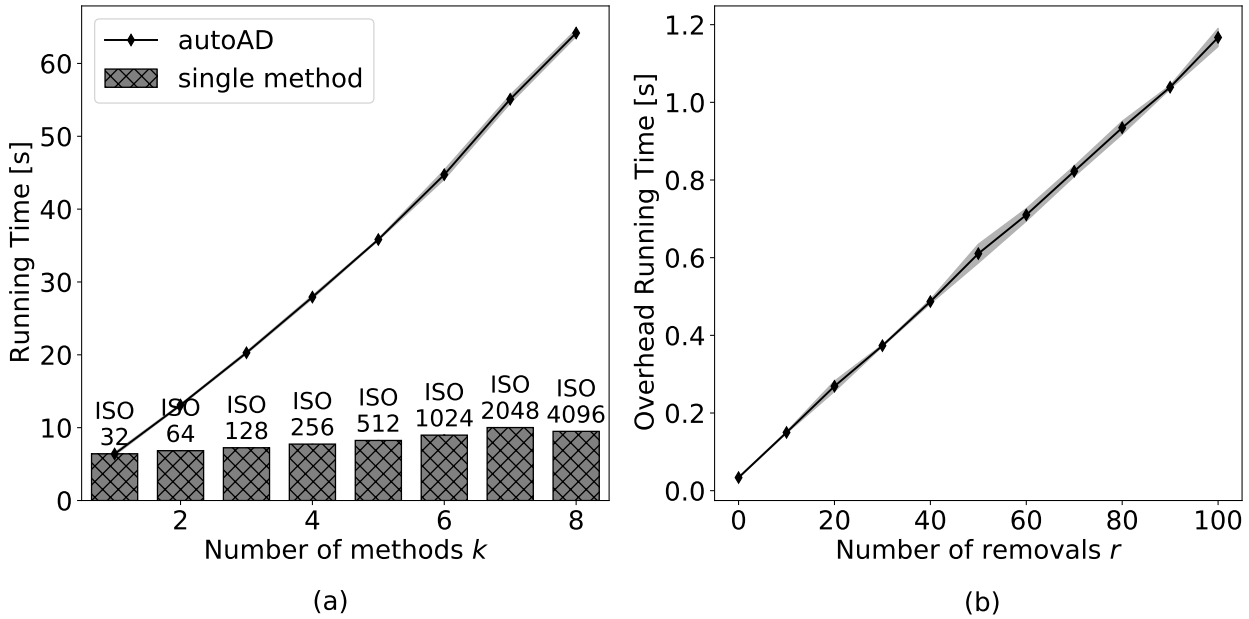


Fig. 6: Running time. (a) the `autoAD` running time together with the running time of each method (bar plot). (b) the running time with different number of removals r.

and intuitive quality measures for each method. Experiments conducted on a various set of datasets show substantial gains of up to 41% in terms of average precision score, while showing linear running time in the number of methods and input data size. We release (anonymously) our Python3 parallel implementation, which runs all different methods in parallel, thereby achieving some significant speedup.

Several improvements can be addressed in the future. We intend to investigate further how to optimally select the set of methods $\mathbb{M}$ and whether we can maintain the average precision performance while reducing the size of the ensemble by removing the ineffective methods (e.g., using only $\{$*ISO@32, ISO@256, ISO@4096*$\}$ or $\{$*RHF@1, RHF@5, RHF@8*$\}$). Besides, more sophisticated unsupervised quality metrics can be explored and further studies on the weighting process can be performed (*e.g.*, instead of the MinMaxScaler, one could only use the first 2/3 best methods and drop the remaining ones).

ACKNOWLEDGEMENTS

This work has been carried out in the frame of a cooperation between Huawei Technologies France SASU and Télécom Paris (Grant no. YBN2018125164).

REFERENCES

[1] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Computing Surveys (CSUR)*, vol. 41, no. 3, pp. 1–58, 2009.

[2] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial Intelligence Review*, vol. 22, no. 2, pp. 85–126, 2004.

[3] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *International Conference on Data Mining*. IEEE, 2008, pp. 413–422.

[4] B. Schölkopf, R. C. Williamson, A. J. Smola, J. Shawe-Taylor, J. C. Platt *et al.*, "Support vector method for novelty detection." in *NIPS*, vol. 12. Citeseer, 1999, pp. 582–588.

[5] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *ACM SIGMOD International Conference on Management of Data*, 2000, pp. 93–104.

[6] A. Putina, M. Sozio, D. Rossi, and J. M. Navarro, "Random histogram forest for unsupervised anomaly detection," in *International Conference on Data Mining (ICDM)*. IEEE, 2020, pp. 1226–1231.

[7] P. B. Brazdil, C. Soares, and J. P. da Costa, "Ranking learning algorithms: Using ibl and meta-learning on accuracy and time results," *Machine Learning*, vol. 50, no. 3, pp. 251–277, 2003.

[8] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning*. Springer, 2019.

[9] M. Bahri, F. Salutari, A. Putina, and M. Sozio, "Automl: state of the art with a focus on anomaly detection, challenges, and research directions," *International Journal of Data Science and Analytics*, pp. 1–14, 2022.

[10] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka," *JMLR*, 2017.

[11] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter, "Auto-sklearn: efficient and robust automated machine learning," in *Automated Machine Learning*, 2019, pp. 113–134.

[12] R. Domingues, M. Filippone, P. Michiardi, and J. Zouaoui, "A comparative evaluation of outlier detection algorithms: Experiments and analyses," *Pattern Recognition*, vol. 74, pp. 406–421, 2018.

[13] A. Emmott, S. Das, T. Dietterich, A. Fern, and W.-K. Wong, "A meta-analysis of the anomaly detection problem," *arXiv preprint arXiv:1503.01158*, 2015.

[14] R. Elshawi, M. Maher, and S. Sakr, "Automated machine learning: State-of-the-art and open challenges," 2019.

[15] M. Feurer and F. Hutter, "Hyperparameter optimization," in *Automated Machine Learning*, 2019, pp. 3–33.

[16] X. He, K. Zhao, and X. Chu, "Automl: A survey of the state-of-the-art," *Knowledge-Based Systems*, 2021.

[17] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle, "The irace package: Iterated racing for automatic algorithm configuration," *Operations Research Perspectives*, 2016.

[18] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *LION*, 2011.

[19] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle, "Paramils: an automatic algorithm configuration framework," *JAIR*, 2009.

[20] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *ACM SIGKDD*, 2019.

[21] B. Komer, J. Bergstra, and C. Eliasmith, "Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn," in *ICML workshop on AutoML*, 2014.

[22] J. Vanschoren, "Meta-learning: A survey," *arXiv preprint arXiv:1810.03548*, 2018.

[23] E. Burnaev, P. Erofeev, and D. Smolyakov, "Model selection for anomaly detection," in *ICMV*, 2015.

[24] Y. Zhao, R. A. Rossi, and L. Akoglu, "Automating outlier detection via meta-learning," *arXiv preprint arXiv:2009.10606*, 2020.

[25] A. Zimek, R. J. Campello, and J. Sander, "Ensembles for unsupervised outlier detection: challenges and research questions a position paper," *SIGKDD*, 2014.

[26] J. Davis and M. Goadrich, "The relationship between precision-recall and roc curves," in *International Conference on Machine Learning*, 2006, pp. 233–240.

[27] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*.

[28] D. Dua and C. Graff, "UCI machine learning repository," 2017.

[29] S. Rayana, "ODDS library [http://odds.cs.stonybrook.edu]," 2016.

[30] M. Goldstein and S. Uchida, "A comparative evaluation of unsupervised anomaly detection algorithms for multivariate data," *PLOS ONE*, 2016.

[31] K. Yamanishi, J.-I. Takeuchi, G. Williams, and P. Milne, "On-line unsupervised outlier detection using finite mixtures with discounting learning algorithms," in *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000.