DeAnomalyzer: Improving Determinism and Consistency in Anomaly Detection Implementations

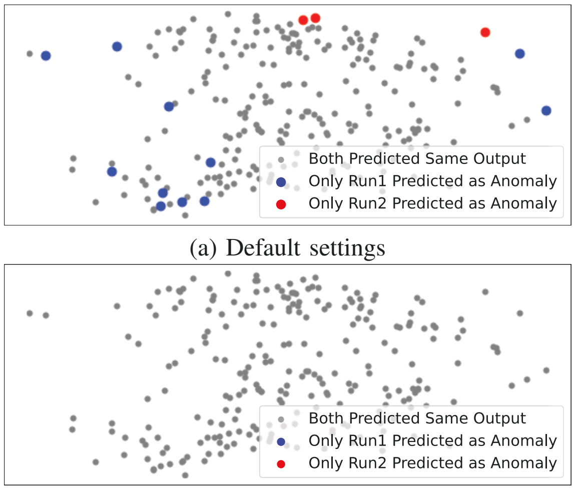
Muyeed Ahmed Iulian Neamtiu
Department of Computer Science, *New Jersey Institute of Technology*, Newark, NJ, USA
{ma234, ineamtiu}@njit.edu

***Abstract*—Anomaly Detection (AD) is a popular unsupervised learning technique, but AD implementations are difficult to test, understand, and ultimately improve. Contributing factors for these difficulties include the lack of a specification to test against, output differences (on the same input) between toolkits that supposedly implement the same AD algorithm, and no linkage between learning parameters and undesirable outcomes. We have implemented DeAnomalyzer, a black-box tool that improves AD reliability by addressing two issues: nondeterminism (wide output variations across repeated runs of the same implementation on the same dataset) and inconsistency (wide output variations between toolkits on the same dataset). Specifically, DeAnomalyzer uses a feedback-directed, gradient descent-like approach to search for toolkit parameter settings that maximize determinism and consistency. DeAnomalyzer can operate in two modes: *univariate*, without ground truth, targeted to general users, and *bivariate*, with ground truth, targeted to algorithm designers and developers. We evaluated DeAnomalyzer on 54 AD datasets and the implementations of four AD algorithms in three popular ML toolkits: MATLAB, R, and Scikit-learn. The evaluation has revealed that DeAnomalyzer is effective at increasing determinism and consistency without sacrificing performance, and can even improve performance.**

***Index Terms*—AI testing, AI reliability, Nondeterminism, Verification, Anomaly Detection, Machine Learning**

I. Introduction

Anomaly Detection (AD) is a widely-used unsupervised learning technique for detecting rare/anomalous items, or data/events that contradict expected behavior. Thanks to AI's growing popularity, AD use is increasing in a wide range of domains, including manufacturing, health, security, and finance. While AD reliability is crucial, verifying or validating the output of an AD algorithm is challenging: there is no specification to check against, and even in scenarios where learning tasks have ground truth, different AD implementations can produce widely diverging outputs on the same input (dataset). In general, AD users should expect (1) *determinism*: a certain AD implementation produces the same output when run repeatedly on the same dataset, and (2) *consistency*: different implementations of the same AD algorithm produce similar outputs on the same input dataset. However, prior work has revealed that popular AD implementations violate both properties [1]. Note that current AD efforts in the ML or Big Data communities are focused on developing new algorithms, optimizing performance, or increasing scalability – little to no research is focused on testing, and improving the reliability of, AD implementations. To address these issues, we designed and implemented DeAnomalyzer: a tool that increases determinism, consistency, and even performance, of AD implementations, without requiring source code, by exploring toolkits' parameter spaces in a gradient descent-like manner. We made DeAnomalyzer available on GitHub. [1]



(b) Improved via our univariate search

Fig. 1. Nondeterminism example: different outputs of two different runs (algorithm: Isolation Forest; toolkit: Sklearn) on dataset vertebral.

To motivate our approach, we illustrate nondeterminism and inconsistency in two popular toolkits, Scikit-learn (Sklearn for short) and MATLAB.

Nondeterminism. First, we illustrate AD nondeterminism on vertebral – an orthopaedic dataset with two classes, normal and abnormal [2]. We ran the Isolation Forest AD algorithm (explained in Section IV) using Sklearn, multiple times, without making any changes in parameters or environment, yet obtained different results in different runs. Figure 1 (a)[2] shows the result of two different runs, "Run 1" and "Run 2". The grey dots represent points where both runs predicted the same output, blue dots are the points that were identified as anomaly only in Run 1, and red dots were the points that were identified as anomaly only in Run 2. Ideally, all the points should be grey (same outcome in both runs), however that is clearly not the case. Among the 240 points, the two runs predicted different outcome for 14 points (cross-run ARI=0.63,

[1]https://github.com/MuyeedAhmed/DeAnomalyzer
[2]For Figures 1 and 2 we used t-SNE [3] to reduce dimensionality of the datasets to 2 for better visualization.

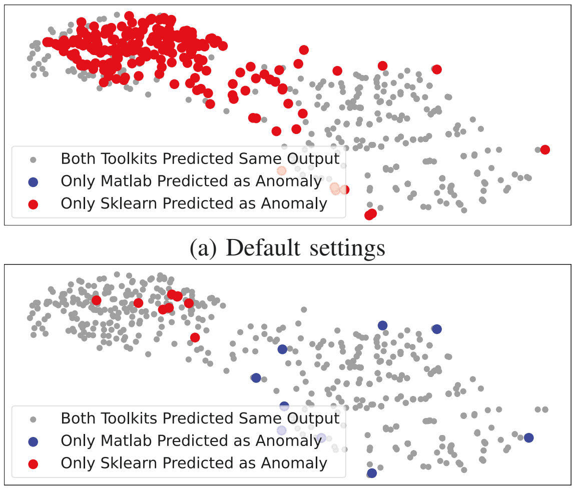


Fig. 2. Inconsistency example: algorithm Isolation Forest on dataset breastw.

far short of the expected ARI=1; Section II-A defines ARI). DEANOMALYZER was able to achieve a cross-run ARI of 1 (eliminating nondeterminism), shown in Figure 1 (b). Moreover, DEANOMALYZER exposed the exact parameter values responsible for nondeterminism, and the values that should be used to eliminate it (e.g., *n_estimators* should be set to 512 instead of the default, 100).

Inconsistency. Second, we ran the Isolation Forest AD algorithm on dataset breastw (Breast Cancer Wisconsin [4], [5]) using the default settings of MATLAB and Sklearn. Figure 2 (a) shows the output, evidencing substantial disagreement between toolkits. The red points represent points identified as anomalies only by Sklearn, and points in grey indicate that both toolkits agreed. While blue points would represent the points only MATLAB identified as anomalies, Figure 2 (a) does not contain such points, as MATLAB only identified a subset of the anomalies exposed by Sklearn.

DEANOMALYZER reduced the inconsistency between the implementations, as illustrated in Figure 2 (b): note the substantial increase of grey dots. DEANOMALYZER exposed the parameter values responsible for the inconsistency, and the values that should be used to reduce it (Matlab's *ContaminationFraction* should be set to a value akin to Sklearn's "auto", i.e., 0.376, instead of the default 0).

Our approach, and DEANOMALYZER's architecture, are discussed in detail in Section III. DEANOMALYZER was designed to operate in a black-box manner, with only knowledge of the toolkit parameter settings (aka "hyperparameters"). DEANOMALYZER uses a feedback-driven gradient descent-like approach to find parameter values that improve determinism or consistency, an approach we name *univariate search*. DEANOMALYZER can also be used to expose the determinism v. performance balance (and respectively, consistency v. performance balance), an approach we name *bivariate search*, because the search moves along two directions. Our experiments found that DEANOMALYZER was able to alleviate nondeterminism in all those 6 implementations that were nondeterministic, and improve consistency for all 11 implementations. For the three strongest nondeterministic algorithms (Matlab/OCSVM, Sklearn/IF, and Matlab/IF), DEANOMALYZER improved ARI from 0.11 to 0.9, 0.78 to 0.94, and 0.84 to 0.94, respectively.

The rest of the paper is structured as follows. Section II introduces definitions and presents the experimental setup. In Sections IV to VII, we present the improvements attained by running DEANOMALYZER on each AD algorithm.

II. DEFINITIONS AND EXPERIMENTAL SETUP

We evaluated our approach on 4 algorithms, as implemented in 3 popular toolkits; we validated the results on 54 AD-specific datasets. This section introduces the evaluation metrics, algorithms, toolkits, and datasets.

A. Definitions and Metrics

Given an unlabeled dataset, AD aims to find abnormal or anomalous points. Therefore, we use two performance metrics: ARI and F1 score. The Adjusted Rand Index (ARI [6]), originating from clustering analysis, measures the similarity between two clustering outputs U and V. In our case, the output represents a partition of a dataset into normal and anomalous points. ARI is versatile as it does not require ground truth, hence can be used to compare AD output obtained from two runs, two toolkits, etc. ARI ranges from -1 to $+1$, where -1 indicates "perfect disagreement" between U and V, while $+1$ indicates the same output or "perfect agreeement" ($U \equiv V$); a score close to 0 means that U and V are random or independent. F1 score measures performance (combining precision and recall of the AD result), and is useful in scenarios, e.g., for AD developers, where ground truth is available and the goal is to improve performance.

B. Algorithms

We explored four AD algorithms, described shortly; three algorithms are prone to both nondeterminism and inconsistency, while one is deterministic, only prone to inconsistency.

Isolation Forest (IF) detects anomalies by isolating anomalous objects. IF uses one or more (potentially infinite number of) estimators. Each estimator selects a random feature from a group of features and then randomly splits a sub-sample of the dataset by selecting a value between the minimum and maximum of that feature [7].

Robust Covariance (RobCov) detects anomalies by drawing a high-dimensional (number of features in a dataset) ellipsoid around the center or "core" of the dataset [8].

Local Outlier Factor (LOF) uses the k-nearest neighbor (kNN [9]) algorithm to measure the distance of each sample in a dataset from its nearest k neighbors and then compares their local density with their neighbors' local density. If local density is significantly lower, that sample is identified as anomaly [10]. LOF is the only deterministic-by-design algorithm among the four.

One Class SVM (OCSVM) uses Support Vector Machines (SVM) to detect anomalies. Note that, to separate classes,

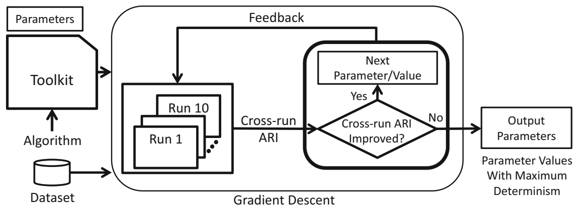


Fig. 3. DeAnomalyzer Determinism improver: univariate search.

SVM maps the points into a high-dimensional feature space and generates class-separating hyperplanes [11]. OCSVM is a simple two-class SVM: the larger class is considered normal, while the smaller class contains anomalies or outliers.

C. Toolkits

We studied AD implementations in three popular toolkits. **Scikit-learn** (Sklearn) has all four aforementioned algorithms built-in. For **MATLAB**, we used the most popular LOF implementation based on GitHub stars [12]; the other algorithms are implemented in official libraries. **R** supports 3 algorithms (IF, LOF, OCSVM) implemented in separate packages [13]–[15]. Therefore, in total we explored 11 implementations.

D. Datasets

The following table summarizes the distribution and characteristics of the 54 datasets we used in our experiments.

	Min	Max	Geometric Mean
Instances (points)	63	7,200	426.7
Features (attributes)	2	64	14.10
# of anomalies	6	2,036	39.88
Anomaly ratio	1.2%	35.9%	9.3%

Out of the 54 datasets, 16 are from Outlier Detection DataSets (ODDS) [16] and the other 38 are from OpenML [17]. On average, the datasets have 426 instances and 14 attributes. The anomaly percentage ranged from 1.2% to 35.9% (typically: 9.3%).

III. DeAnomalyzer Architecture

We designed DeAnomalyzer to improve the determinism of a given AD toolkit, and the consistency between two given AD toolkits. Moreover, when ground truth is provided, DeAnomalyzer incorporates mechanisms to preserve, and even increase, performance.

A. Nondeterminism

To reduce nondeterminism, DeAnomalyzer supports two strategies: *univariate search* and *bivariate search*. Univariate search optimizes for a single output variable, determinism, and is applicable in scenarios where repeatability or reproducibility are key; this strategy does not take into account performance. In contrast, bivariate search optimizes for both determinism and performance; this strategy is applicable in, for example, validation settings, where ground truth is available.

```
procedure DeAnomalyzer_Univariate:
Input: dataset $d$, implementation $t$, $ParameterValues$
Output: $OptimumParamSetting$
$ParameterValues$=$m$ parameters, each with $n$ values
for all parameters $p_i$ where $i = 1, ..., m$ {
  $p_iValues = \{p_{i1}, p_{i2}, \ldots p_{in}\}$
  $p_{ij}$ = $default$ from $p_iValues$
  $CurrentValues$ = set $p_i$ to $p_{ij}$ in $ParameterValues$
  do {
    // run executes $t$ 10 times on $d$ and returns the mean ARI
    $ARIScore$ = run($t$, $d$, $CurrentValues$)
    Append $p_{ij}$ to $OptimumParamSetting$
    // Update $CurrentValues$
    if $p_{ij+1}$ performs better than $p_{ij}$
      $p_i = p_{ij+1}$
    else
      $p_i = p_{ij-1}$
  } while ($ARIScore$ increases)
}
return $OptimumParamSetting$
```

Fig. 4. Univariate search pseudocode.

1) Univariate Search: Figure 3 shows an overview of univariate search approach: given a toolkit, the parameter space is explored, informed via feedback on determinism values (cross-run ARI), to find values that maximize determinism.

Figure 4 shows the pseudocode for univariate search. DeAnomalyzer takes as input the dataset d, an algorithm implementation t (e.g., Sklearn/IF), and the set of parameters $ParameterValues$. For each parameter p_i, we have a list of possible values $[p_{i1}, p_{i2}, ...]$.[3] DeAnomalyzer first starts with a parameter p_i and then runs t on the dataset using the default value p_{ij} for p_i (all other parameters are set to default as well) 10 times, and computes the mean cross-run ARI. Note that a high ARI value indicates determinism across the 10 runs, while a low ARI value indicates nondeterminism. Then DeAnomalyzer selects the next value p_{ij+1} from the possible value list of p_i and runs t again 10 times. If the cross-run ARI score is better than the score with default settings, DeAnomalyzer explores in that direction (the next value for p_i being p_{ij+2}) and will continue doing so as long as the cross-run ARI increases. However, if the cross-run ARI with p_{ij+1} is worse than the default value's score, DeAnomalyzer will start exploring the values that precede the default (i.e., $p_{ij-1}, p_{ij-2}, ..$). Before moving on to the next parameter, DeAnomalyzer selects the parameter value with the maximum determinism as the new default for p_i and also stores the "winner" in $OptimumParamSetting$. DeAnomalyzer then follows the same procedure for exploring other parameters of t. Finally, after going through all parameters, $OptimumParamSetting$ will have the settings, e.g., parameter values, that represent a nondeterminism minimum (i.e., determinism maximum) for t on dataset d.

For some parameters the default value is dynamic (i.e., depends on the dataset). For example, the default value for *max_samples* in Sklearn/IF is $min(256, \#of samples)$, meaning the implementation will take 256 samples from the

[3]We had a few cases where parameter values were defined as continuous values over a range, rather than a list of discrete values. In those cases we created a 10-value list manually.

dataset to train an estimator if the dataset has more than 256 points, otherwise it will take all the points. Therefore, if the dataset contains 1000 points, the default *max_samples* should come after 0.2 (20% of the points) and before 0.3 in the list ($[0.1, 0.2, default, 0.3, \ldots]$) as now the fraction of points to be used in each estimator is 0.256 (256/1000).

We illustrate univariate search on Isolation Forest, toolkit Sklearn. As mentioned previously, *max_samples* controls the fraction of points to use to build a single estimator. The range of *max_samples* is $(0, 1]$ so as list of values we divided this range into 10 discrete values ($[0.1, 0.2, \ldots, 1.0]$). Assuming a default value *max_samples=0.5*, DEANOMALYZER runs Sklearn/IF 10 times on the dataset using 0.5 as *max_samples* and computes the cross-run ARI. DEANOMALYZER then moves on to the next value, *max_samples=0.6*, runs Sklearn/IF 10 times, and obtains a new cross-run ARI. If the new cross-run ARI is better, DEANOMALYZER moves to the next value, *max_samples=0.7* and will continue doing so until the ARI starts to decrease or we exhausted the *max_samples*'s set of values. However if we obtained a lower ARI score, DEANOMALYZER will select the *max_samples* value of 0.4 and potentially continue exploring downwards (0.3, 0.2, …) until a decrease in ARI score is observed. Assuming, for example, that *max_samples=0.3* yields the maximum cross-run ARI, DEANOMALYZER sets this value as the new default and adds 0.3 in our $OptimumParamSetting$, i.e., output parameters. Next, DEANOMALYZER explores the remaining parameters of Sklearn/IF following the same strategy. For unbounded parameters we manually set a realistic upper bound (e.g., for *n_estimators* we set the upper bound to 512).

Limitation: local v. global minima. As typical in gradient descent, our approach assumes the function to be optimized is convex, hence could discover local, rather than global, nondeterminism minima. This is a deliberate trade-off to keep exploration time tractable, as in the worst case the number of runs is still linear in the number of parameter values. In contrast, the number of runs in grid search will be exponential.

Figure 5 shows two examples of univariate search. In Figure 5 (a), we used Matlab/OCSVM on dataset analcatdata_apnea3 and in Figure 5 (b) we used Sklearn/IF on dataset ar1. In both examples, we can observe a steady rise in cross-run ARI with each step. In Figure 5 (a), with default settings the cross-run ARI was 0.008; *note that cross-run ARI close to 0 indicates outputs so different across runs as to appear unrelated*. First, DEANOMALYZER explored the parameter *ContaminationFraction* (CF) but changing the default value of 0.1 in either direction did not improve the ARI score. Then, DEANOMALYZER moved on to the next parameter, *KernelScale (KS)* ; the default value for *KS* is 1 and setting it to "auto" (which invokes a heuristic procedure, i.e., the algorithm will choose the value automatically) increased the cross-run ARI by almost 0.5. After exploring the remaining parameters (*Lmda*, *SD*, *BT*), DEANOMALYZER achieved a cross-run ARI of **0.67**, which is a substantial improvement from the default, 0.008. We also see a minor rise in F1 score, from 0.12 to 0.15. In the Figure 5 (b) example, DEANOMALYZER found an optimal value for *n_e* in three steps, then optimal values for *w_s* and *m_s*). In the end, DEANOMALYZER increased cross-run ARI from 0.84 to 0.93, with a minor decline in F1 score (from 0.287 to 0.277).

2) Bivariate Search: Figure 6 shows an overview of the bivariate search approach: given a toolkit, the parameter space is explored, informed via a feedback loop on both determinism and performance values. Here, DEANOMALYZER explores the determinism v. performance space. While ideally we wish to improve both, this might not be possible in practice, hence DEANOMALYZER allows users to quantify, say, the $x\%$ performance loss incurred by a $y\%$ gain in determinism.

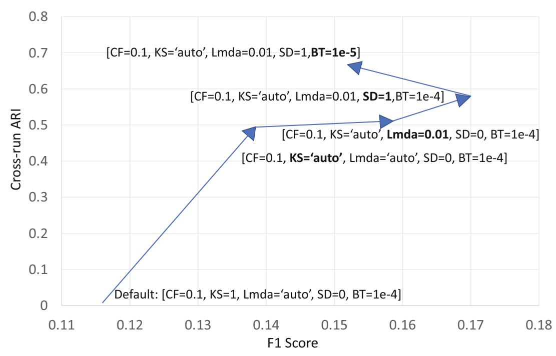
Bivariate search is similar to univariate search, but considers performance. When comparing the result gathered by default p_{ij} and p_{ij+1}, we check both the cross-run ARI and F1 score, and move onto p_{ij+2} if the cross-run ARI increased and F1 score did not decrease. However if the F1 score decreases, we move on to p_{ij-1} even when cross-run ARI increases.

Figure 7 shows two examples of bivariate search. For the two examples we used the same datasets and tool/algorithm combinations as in Figure 5. In Figure 5 (a) we see an increase of cross-run ARI after changing the value of *BetaTolerance (BT)* from the default 10^{-4} to 10^{-5}. While in univariate search we explored that path, in bivariate search we did not, as it reduces performance, as can be seen in Figure 7 (a). In this example we do see an increase of both cross-run ARI (0.58) and F1 score (0.17). In Figure 5 (b), taking the path to *n_estimators* (n_e)=512 from *n_estimators*=256 would reduce performance, hence we stop following that path. The final result is no decrease in performance (0.30), while improving cross-run ARI (0.90).

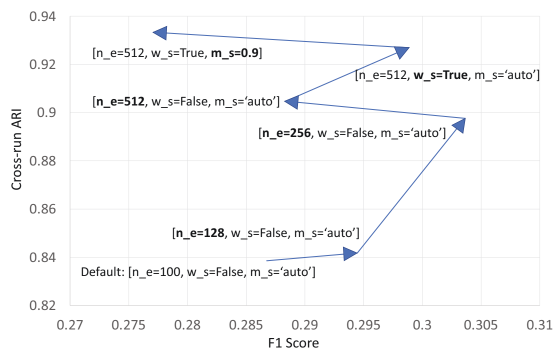
B. Inconsistency

To reduce inconsistency we adapted univariate and bivariate search to operate on two toolkits. We had to address the challenge where for a certain algorithm, the number of parameters differed between toolkits, e.g., MATLAB/RobCov has 9 parameters but Sklearn only has 4. DEANOMALYZER addresses this by exploring all available parameters for a certain toolkit. Figure 8 shows an overview of our approach.

1) Univariate Search: In univariate search targeting nondeterminism, after moving to a new parameter value we calculated the cross-run ARI. In univariate search targeting inconsistency, we "nest" one toolkit t_b inside the other toolkit t_a. Specifically, we first choose a parameter p_a from toolkit t_a and start exploring that parameter. We choose the pa_i value of parameter p_a and then choose the pb_i value from parameter p_b of toolkit t_b and check the mutual ARI between toolkits t_a and t_b. Then we move on to pb_{i+1} in t_b and similar to univariate search targeting nondeterminism we check and move to the next step. Note that instead of cross-run ARI, here we check the mutual ARI of the two tools' output. While exploring toolkit t_b we continue to update the default value with the new default value for each parameter. After exploring toolkit t_b completely (all parameters), we change the p_a parameter value of toolkit a to pa_{i+1} and continue with the same process



(a) MATLAB/OCSVM, dataset analcatdata_apnea3



(b) Sklearn/IF, dataset ar1

Fig. 5. DEANOMALYZER univariate search example: reducing nondeterminism.

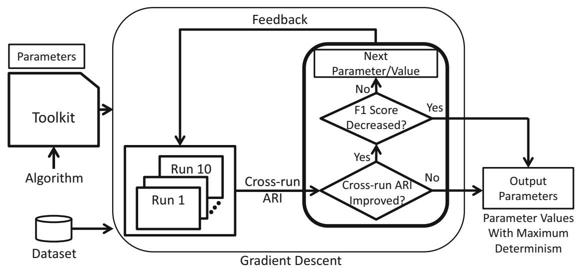


Fig. 6. DEANOMALYZER determinism improver: bivariate search.

again (run toolkit t_b). We will continue doing this for all the parameters of toolkit t_a. Finally, we will have two sets of parameter settings, for t_a and t_b respectively, that represent a consistency maximum for the given dataset.

For example, Sklearn/IF has a parameter *n_estimators* whose default value is 100. When aiming to improve the consistency between Sklearn/IF and MATLAB/IF, DEANOMALYZER checks the mutual ARI of Sklearn and MATLAB with this setting and continues exploring MATLAB, starting from parameter *ContaminationFraction* and other subsequent parameters as long as the mutual ARI increases. Next, DEANOMALYZER moves back to Sklearn and changes the value of *n_estimators* to the next option (in this case, 128) and finds the setting in MATLAB that achieves the highest agreement with (i.e., best mutual ARI) Sklearn's current setting. DEANOMALYZER continues this approach until all the parameters of Sklearn have been explored.

Figure 9 shows how univariate search reduces inconsistency. With default settings, the mutual ARI between the two toolkits is low, 0.070. The first change, the *Method* parameter in MATLAB flip from "fcmd" to "ogk", brought the mutual ARI up to 0.588. After exploring all the parameters in MATLAB once, DEANOMALYZER went back to Sklearn and changed *assume_centered* from "false" to "true", which increased mutual ARI slightly from 0.686 to 0.710. In the next few iterations, MATLAB did not improve the mutual ARI. After changing the Sklearn parameter *contamination* to "IF" we saw another small increase. Finally, the mutual ARI increased to **0.866**.

2) Bivariate Search: For bivariate search we follow the same procedures, but also optimizing for F1 score. Additionally as we already have the ground truth, we nest our tools in such a way that the toolkit with the worse outcome (lower F1 score) tries to get closer to the toolkit with better outcome. For example, for algorithm Isolation Forest, the average F1 score in MATLAB with default settings was 0.26 and in Sklearn it was 0.294. Therefore we mirrored Sklearn's parameter exploration in MATLAB (i.e., after changing a parameter's value in Sklearn, all the parameters will be explored in MATLAB) in order to increase the performance of MATLAB, resulting in an increased mutual ARI. After the bivariate search, the F1 score of MATLAB was 0.326, Sklearn's was 0.322, while the mutual ARI increased from 0.551 to 0.598.

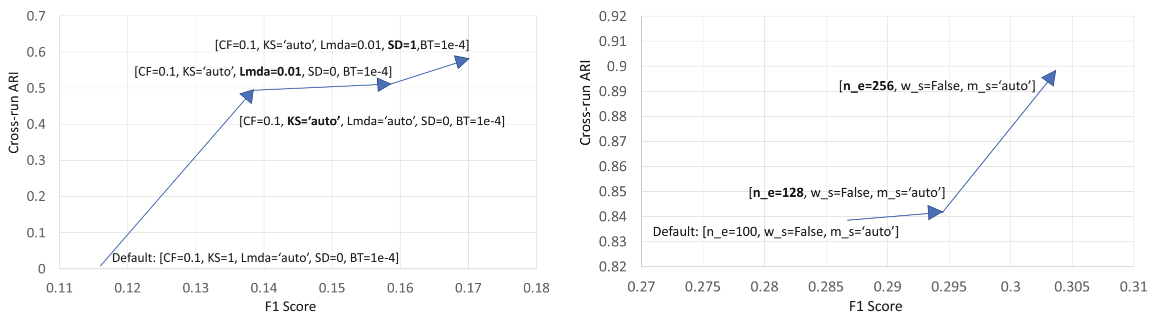
We now present the evaluation results for each algorithm.

IV. ISOLATION FOREST

A. Nondeterminism

Isolation Forest implementations are nondeterministic in 2 of the 3 toolkits (only R was deterministic [1]); specifically, Sklearn and MATLAB were nondeterministic for more than 90% of datasets. We now discuss how DEANOMALYZER reduces nondeterminism without sacrificing performance.

1) Sklearn: DEANOMALYZER revealed that *n_estimators* is one of the most influential parameters for reducing nondeterminism (*n_estimators* is the number of base estimators the algorithm can use to predict anomalies). The default setting of *n_estimators* was 100, with possible values $\{2, 4, 8, ..., 512\}$; DEANOMALYZER revealed that increasing the value can reduce nondeterminism. DEANOMALYZER runs with different values of this parameter on each dataset 10 times and produces cross-run ARI scores (Section III). Table I shows the results of a Mann-Whitney U test [18] for each pair [(64, 100), (64, 128), (64, 256),...] across all datasets. The strong significance levels (< 0.05 for all but one) essentially indicate that *each different value of the parameter produces very different levels of determinism* (cross-run ARI). The other influential parameter is *max_samples* (which determines the fraction of points to be used in each estimator). Table II shows other IF parameters with their AIC value (Akaike information criterion [19]) – lower values indicate higher relevance. These values essentially quantify the importance of each parameter.



(a) MATLAB/OCSVM, dataset analcatdata_apnea3

(b) Sklearn/IF, dataset ar1

Fig. 7. DEANOMALYZER bivariate search example: reducing nondeterminism.

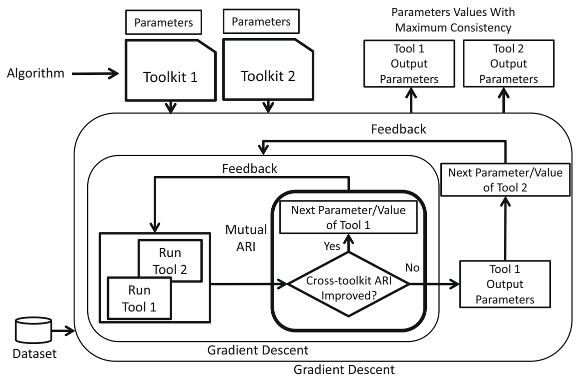


Fig. 8. DEANOMALYZER consistency improver: univariate search.

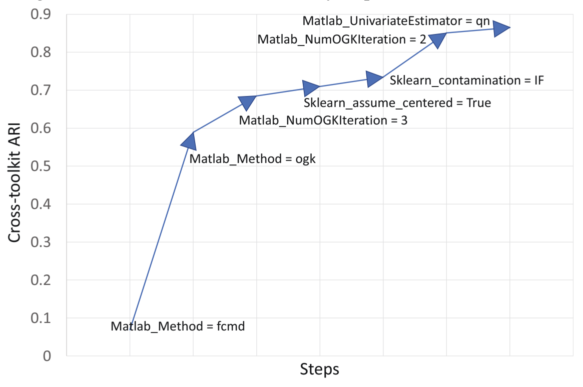


Fig. 9. DEANOMALYZER improving consistency in dataset vertebral using Robust Covariance algorithm in Sklearn and MATLAB.

Table III shows how DEANOMALYZER performed on Isolation Forest algorithm in Sklearn. In univariate search we see a high improvement in determinism (19.8%, i.e., from 0.783 to 0.938) with the mean performance virtually identical to the default setting. In bivariate search we see a slight increase in determinism (from 0.783 to 0.84) without compromising performance. Univariate search improved determinism in 53 out of 54 datasets; one dataset's determinism remained unchanged. In case of performance we do not see any mean difference with the default, but in 30 datasets we observed an increase in F1 score while in 23 datasets we observed a decrease in F1 score (performance on one dataset remained unaffected). In bivariate search, out of 54 datasets, we see that 42 have

TABLE I
IF/SKLEARN N_ESTIMATORS: U TEST SCORE OF CROSS-RUN ARI

	100	**128**	**256**	**512**
64	0.06573	0.00362	4.43E-06	2.57E-10
100		0.26738	0.00331	2.84E-07
128			0.02761	6.57E-06
256				0.00943

TABLE II
IF/SKLEARN: AIC VALUES, IN DECREASING ORDER OF IMPORTANCE

max_samples	**n_estimators**	**max_features**	**bootstrap**	**n_jobs**	**warm_start**
35	64	138	171	171	171

TABLE III
IF: IMPACT OF DEANOMALYZER

	Config.	**Determinism**			**Performance**		
			#Datasets			#Datasets	
		Mean	Better	Worse	Mean	Better	Worse
Sklearn	Default	0.783	-	-	0.294	-	-
	Univariate	0.938	53	0	0.294	30	23
	Bivariate	0.840	42	0	0.308	42	0
MATLAB	Default	0.839	-	-	0.260	-	-
	Univariate	0.943	53	0	0.245	26	25
	Bivariate	0.881	40	0	0.298	40	0
R	Default	0.983	-	-	0.260	-	-
	Univariate	0.992	35	0	0.279	20	15
	Bivariate	0.989	23	0	0.291	23	0

better determinism and performance than the default.

2) MATLAB: While the MATLAB implementation of the IF algorithm is nondeterministic, a default parameter setting renders it de facto deterministic in an unexpected way. Specifically, the default *ContaminationFraction* is 0, meaning the implementation will label 0% of the points as anomaly – a questionable default value yielding a deterministic "no anomalies" outcome. Hence, though deterministic, this outcome is not desirable due to false negatives; in other words, if the dataset contains outliers, the F1 score will be 0. Typically users set the value of *ContaminationFraction* to 0.05 or 0.1 meaning the user expects the dataset to have 5% or 10% outliers [20], [21]. MATLAB does not offer a strategy for predicting an appropriate value for this parameter. Therefore, we created a set of possible *ContaminationFraction* values based on the anomaly percentage predictions made by Sklearn/IF and Sklearn/LOF.

As comparing it with the default *ContaminationFraction* would be pointless, we compared the custom settings with a

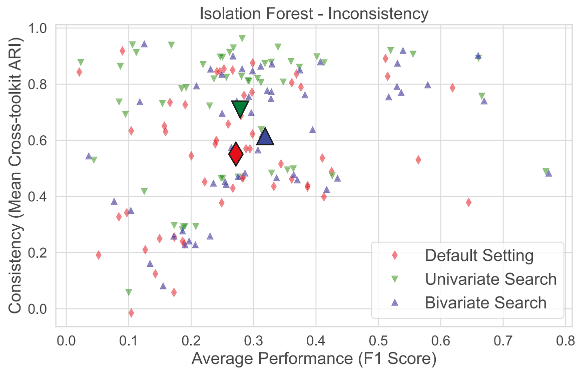


Fig. 10. Isolation Forest: Performance v. Consistency before (default) and after DeAnomalyzer's search. The three large points are mean F1 score and ARI across all datasets.

modified default setting, where the *ContaminationFraction* is set to 0.1. Univariate search increased cross-run ARI by 12.4% but led to a 5.8% loss in F1 score. However with bivariate search, we see both a 5% increase in cross-run ARI and a 14.6% increase in F1 score.

3) R: R's implementation of Isolation Forest uses the same random seed in each run, yielding deterministic results. However this behavior is vulnerable to changes in seed values. We experimented with different seed values in different runs and observed nondeterminism for 43 out of the 54 datasets. Although nondeterministic, the cross-run ARI was very high, averaging 0.98. This is mainly due to its default value of *ntrees* set to 500; note that *ntrees* in R/IF has similar semantics to *n_estimators* in Sklearn/IF. The average F1 score in default setup is 0.26. With DeAnomalyzer in univariate search, determinism raised to 0.992; in bivariate search, DeAnomalyzer increased accuracy by 11.9%.

B. Inconsistency

Figure 10 shows the gains thanks to DeAnomalyzer's bivariate and univariate search. The x-axis is the average F1 score for each toolkit, while the y-axis is the average mutual ARI (Sklearn v. R, R v. MATLAB, and Sklearn v. MATLAB). Each point represents a dataset, with default in red, univariate in green, and bivariate in blue. The enlarged points in the figure represent the mean of all datasets (performance and consistency) in each setting. In both univariate and bivariate search we see an increase *in both performance and consistency* – note how the green points have shifted up (higher consistency) and the blue points have shifted up and right (better consistency and performance). DeAnomalyzer has revealed that, for most datasets, a high increase in consistency can be achieved by matching values of parameters *ContaminationFraction* and *sample_size* between toolkits.

V. Robust Covariance

Robust Covariance, supported by MATLAB and Sklearn, has nondeterministic behavior in both implementations; the two implementations are inconsistent as well.

A. Nondeterminism

We discuss DeAnomalyzer's outcome on each toolkit; the results are shown in Table IV.

1) Sklearn: Unlike Sklearn/IF and Sklearn/LOF, Sklearn/RobCov does not follow any algorithm or method to predict the number of outliers in a dataset; instead a default value of 0.1 is used (meaning 10% of the points will be labeled as outliers).

TABLE IV
RobCov: Impact of DeAnomalyzer

	Config.	**Determinism**			**Performance**		
			#Datasets			#Datasets	
		Mean	Better	Worse	Mean	Better	Worse
Sklearn	Default	0.87	-	-	0.25	-	-
	Univariate	0.93	37	0	0.26	21	14
	Bivariate	0.91	29	0	0.28	29	0
MATLAB	Default	0.978	-	-	0.077	-	-
	Univariate	1.0	12	0	0.120	14	10
	Bivariate	0.984	11	0	0.147	22	0

Both univariate and bivariate search performed better than the default on average, with many datasets achieving a cross-run ARI of 1 for both univariate and bivariate search. From Table IV we can see that determinism increased in both univariate (6.9%) and bivariate (4.6%) search without losing performance. In univariate search, out of 54 datasets we see a decrease in performance in 14, but overall we gained 4% in performance (mean F1 score increase across all datasets).

2) MATLAB: While MATLAB has good determinism with default settings, DeAnomalyzer's univariate search made performance completely deterministic (mean cross-run ARI increased from 0.978 to 1). However with default settings, the implementation has a very low F1 score (on average 0.077), as for 38 datasets, the algorithm failed to detect any anomalies. DeAnomalyzer managed to increase the overall performance to 0.147.

B. Inconsistency

The MATLAB and Sklearn implementations of RobCov were very inconsistent (mean mutual ARI was 0.26); while DeAnomalyzer managed to reduce inconsistency substantially, it did not eliminate it altogether.

Figure 11 shows the mutual ARI of the univariate search and bivariate search with default setting between the two implementations. With univariate search, DeAnomalyzer increased mutual ARI from 0.26 to 0.7 and for the bivariate search from 0.26 to 0.5, as seen in Table V. In terms of performance, we only see a small decrease in Sklearn (6%) with univariate search.

VI. Local Outlier Factor

LOF is by design a deterministic algorithm, so in all toolkits we saw deterministic output for all datasets. Nevertheless, the output was inconsistent across toolkits, which DeAnomalyzer successfully reduced.

Recall that LOF uses a local density metric to identify outliers (Section II-B). In practice, the metric is Local Reachability Density (LRD), an inverse density measure (i.e., a high

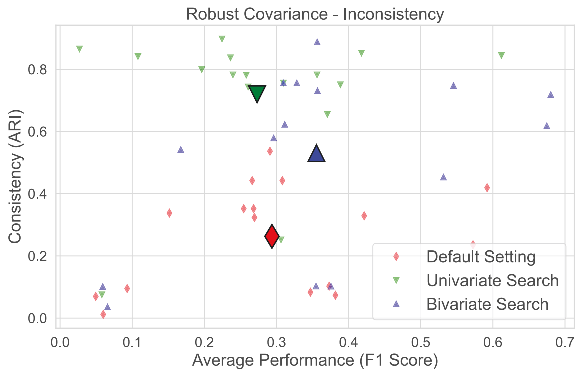


Fig. 11. Robust Covariance: Performance v. Consistency before (default) and after running DeAnomalyzer.

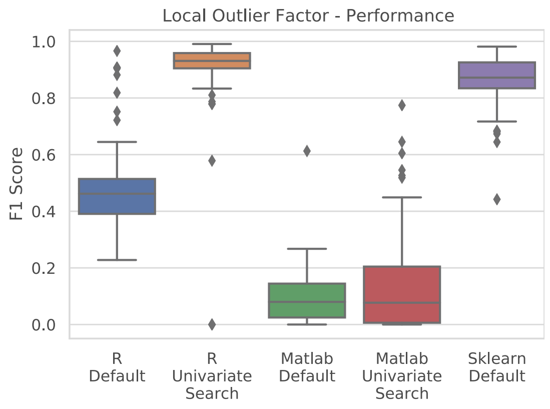


Fig. 12. Local Outlier Factor: performance comparison.

TABLE V
Robust Covariance: Impact of DeAnomalyzer

		Default	Univariate	Bivariate
Mutual ARI (Sklearn v. MATLAB)	Mean	0.26	0.70	0.50
	#Datasets Better	-	16	16
	#Datasets Worse	-	0	0
Performance Sklearn	Mean	0.33	0.31	0.40
	#Datasets Better	-	8	12
	#Datasets Worse	-	6	0
Performance MATLAB	Mean	0.26	0.29	0.30
	#Datasets Better	-	10	14
	#Datasets Worse	-	5	0

TABLE VI
Local Outlier Factor: Impact of DeAnomalyzer

	Sklearn v. R	Sklearn v. MATLAB	R v. MATLAB
Default	-0.012	0.132	-0.001
Univariate	0.913	0.923	0.892

LRD value indicates the point is in a non-dense region, hence has a high probability of being an outlier). Implementations use a *threshold* LRD value to label the point normal (below threshold) or outlier (above threshold). MATLAB and R use a predefined *threshold* value: 1 in R and 2 in MATLAB. This *threshold* produces poor performance in MATLAB and R, which DeAnomalyzer successfully addressed.

We plot the DeAnomalyzer performance improvements in Figure 12. The 'default' boxplots show the performance (statistics across all datasets) for each toolkit. Note the average F1 scores: 0.097 in MATLAB and 0.49 in R. In contrast, Sklearn's performance was significantly better (average F1 score: 0.87). Sklearn operates differently: by default (*contamination* set to "auto"), Sklearn sets the *threshold* to 1.5 and it produces a binary (normal/outlier) output. With different *contamination* values, Sklearn calculates the appropriate *threshold*. Another important parameter is the *minPts* (in R) or *k* (in Sklearn and MATLAB) – the number of nearest neighbors of a point. By default, the parameter is set to 20 in Sklearn/MATLAB and to 5 in R. DeAnomalyzer was able to increase the F1 score of R substantially (from 0.49 to 0.89); for MATLAB, the improvement was modest (from 0.097 to 0.237, a 0.14 gain). For both R and MATLAB we set the *threshold* to match the percentage of outliers identified by Sklearn's auto strategy (in other words, the output of all 3 implementations will have the same number of anomalies). This moved MATLAB and R's *threshold* and *minPts* values close to Sklearn' default values.

We present the results, substantial increase in consistency among all three toolkits, in Table VI. In all three cases after running DeAnomalyzer, the cross-toolkit ARI moved from roughly 0 (no similarity between outputs) to 0.9 (strong agreement between outputs), a significant increase. Note that for LOF, both univariate and bivariate search achieved the same result, so we omit the bivariate results from the table.

VII. One Class SVM

First, we show how DeAnomalyzer increases MATLAB/OCSVM's determinism, and then show how DeAnomalyzer improves consistency between toolkits.

A. *Nondeterminism*

Our experiments indicated that MATLAB has high nondeterminism and low performance. We show the default settings' outcomes in Figure 13: as the red points indicate, average determinism (ARI) was 0.105 and average F1 score was 0.127. DeAnomalyzer was able to increase both determinism and performance significantly. Table VII shows the average cross-run ARI of both univariate and bivariate search rise substantially (from 0.105 to 0.896 and 0.778, respectively) and the F1 score doubles (from 0.127 to 0.258 and 0.288, respectively).

TABLE VII
MATLAB/OCSVM: Impact of DeAnomalyzer

Configs	**Determinism**			**Performance**		
		#Datasets			#Datasets	
	Mean	Better	Worse	Mean	Better	Worse
Default	0.105	-	-	0.127	-	-
Univariate	0.896	54	0	0.258	44	10
Bivariate	0.778	51	0	0.288	50	0

B. *Inconsistency*

The implementations of OCSVM are highly inconsistent. In default settings we found the mutual ARI to be below 0.1 for MATLAB v. R and MATLAB v. Sklearn, whereas for Sklearn v. R it was 0.18. When DeAnomalyzer "imposed" Sklearn's

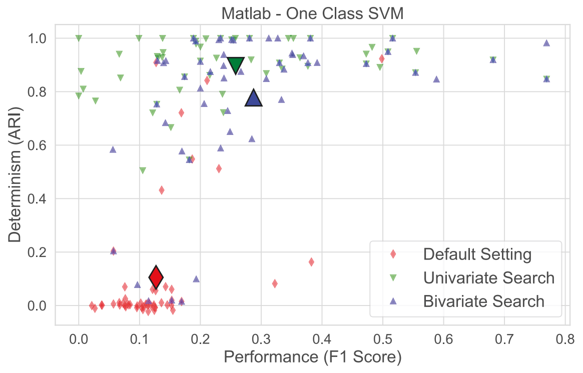


Fig. 13. MATLAB/OCSVM: Performance v. Determinism before (default) and after running DEANOMALYZER.

nu parameter value onto R and MATLAB (in MATLAB that parameter's name is *ContaminationFactor*), a significantly high consistency was achieved. Specifically, for MATLAB v. R consistency improved from 0.007 to 0.378; for MATLAB v. Sklearn, consistency increased from 0.02 to 0.477; for Sklearn v. R, the increase was from 0.18 to 0.39.

VIII. Related Work

We found little prior work on AD reliability, and no attempt to improve AD implementations' reliability.

Ahmed and Neamtiu [1] measured nondeterminism and inconsistency among AD implementations of 4 popular AD algorithms; they found that more than half of the implementations are nondeterministic and all implementation pairs are inconsistent. Soenen et al. [22] studied the effect of hyperparameter tuning between default and maximum performance within the same toolkit, and proposed a strategy to tune parameters for a given dataset. However these efforts have not attempted to alleviate nondeterminism or inconsistency.

Nondeterminism and inconsistency have been studied in the context of clustering algorithms. Several studies have exposed and quantified the issues [23]–[25] without providing a solution. Yin et al. [26] took a manual white-box approach for exposing the root causes of, and reducing, nondeterminism and inconsistency for clustering implementations. Their approach is not applicable here, as AD and clustering solve different problems; in addition, their white-box approach was focused on specific issues in specific implementations, rather than being a black-box optimization tool.

IX. Conclusions

Given the rising popularity of Anomaly Detection and evidence that AD can produce nondeterministic and inconsistent results, there is a need for approaches that allow testing, and increasing the reliability of, AD implementations. Our approach DEANOMALYZER explores AD toolkits' parameter spaces to reduce nondeterminism and inconsistency. An evaluation on 11 AD implementations has confirmed that parameter-based optimization is an effective approach, and has established that DEANOMALYZER is effective at achieving higher determinism and consistency on given AD implementations.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant No. CCF-2007730.

References

[1] M. Ahmed and I. Neamtiu, "Anomalous anomaly detection," in *2022 IEEE International Conference On Artificial Intelligence Testing (AITest)*, 2022, pp. 1–6.
[2] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml
[3] L. Van der Maaten and G. Hinton, "Visualizing non-metric similarities in multiple maps," *Machine learning*, vol. 87, no. 1, pp. 33–55, 2012.
[4] W. N. Street, W. H. Wolberg, and O. L. Mangasarian, "Nuclear feature extraction for breast tumor diagnosis," in *Biomedical Image Processing and Biomedical Visualization*, ser. SPIE, R. S. Acharya and D. B. Goldgof, Eds., vol. 1905, Jul. 1993, pp. 861–870.
[5] W. H. Wolberg and O. L. Mangasarian, "Multisurface method of pattern separation for medical diagnosis applied to breast cytology." *Proceedings of the National Academy of Sciences*, vol. 87, no. 23, pp. 9193–9196, 1990. [Online]. Available: https://www.pnas.org/content/87/23/9193
[6] L. Hubert and P. Arabie, "Comparing partitions," *Journal of Classification*, vol. 2, pp. 193–218, 02 1985.
[7] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *ICDM*. IEEE, 2008, pp. 413–422.
[8] P. J. Rousseeuw and K. V. Driessen, "A fast algorithm for the minimum covariance determinant estimator," *Technometrics*, vol. 41, no. 3, pp. 212–223, 1999.
[9] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27, 1967.
[10] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *ACM SIGMOD*, 2000, pp. 93–104.
[11] K. Heller, K. Svore, A. D. Keromytis, and S. Stolfo, "One class support vector machines for detecting anomalous windows registry accesses," 2003.
[12] "Anomaly detection toolbox." [Online]. Available: https://github.com/dsmi-lab-ntust/AnomalyDetectionToolbox/tree/master/Algorithms/distributionBased/LOF
[13] David-Cortes, "David-cortes/isotree: (python, r, c/c++) isolation forest and variations such as sciforest and eif, with some additions (outlier detection + similarity + na imputation)." [Online]. Available: https://github.com/david-cortes/isotree
[14] "Local outlier factor score." [Online]. Available: https://search.r-project.org/CRAN/refmans/dbscan/html/lof.html
[15] "Svm: Support vector machines." [Online]. Available: https://www.rdocumentation.org/packages/e1071/versions/1.7-9/topics/svm
[16] "ODDS," April 2022, http://odds.cs.stonybrook.edu/.
[17] "OpenML," April 2022, https://www.openml.org/.
[18] P. E. McKnight and J. Najab, "Mann-whitney u test," *The Corsini encyclopedia of psychology*, pp. 1–1, 2010.
[19] Y. Sakamoto, M. Ishiguro, and G. Kitagawa, "Akaike information criterion statistics," *Dordrecht, The Netherlands: D. Reidel*, vol. 81, no. 10.5555, p. 26853, 1986.
[20] "Fit isolation forest for anomaly detection - MATLAB," December 2022, https://www.mathworks.com/help/stats/iforest.html.
[21] "Code Generation for Anomaly Detection," December 2022, https://www.mathworks.com/help/stats/code-generation-for-anomaly-detection.html.
[22] J. Soenen, K. Leuven, E. V. Wolputte, L. Perini, V. Vercruyssen, W. Meert, J. Davis, and H. Blockeel, "The effect of hyperparameter tuning on the comparative evaluation of unsupervised anomaly detection methods," ser. ODD '21: 6th Outlier Detection and Description Workshop, 2021.
[23] V. Musco, X. Yin, and I. Neamtiu, "Smokeout: An approach for testing clustering implementations," in *ICST 2019*, April 2019.
[24] X. Yin, V. Musco, I. Neamtiu, and U. Roshan, "Statistically rigorous testing of clustering implementations," in *AITEST 2019*, April 2019.
[25] S. Rahaman, R. Samuel, and I. Neamtiu, "Quantifying nondeterminism and inconsistency in self-organizing map implementations," in *IEEE AITest*, 2021, pp. 85–92.
[26] X. Yin, I. Neamtiu, S. Patil, and S. T. Andrews, "Implementation-induced inconsistency and nondeterminism in deterministic clustering algorithms," in *ICST 2020*, October 2020.