Implementation-induced Inconsistency and Nondeterminism in Deterministic Clustering Algorithms

Xin Yin, Iulian Neamtiu, Saketan Patil, Sean T. Andrews
Department of Computer Science, *New Jersey Institute of Technology*, Newark, NJ, USA
{xy258, ineamtiu, sap254, sean.t.andrews}@njit.edu

***Abstract*—A deterministic clustering algorithm is designed to always produce the same clustering solution on a given input. Therefore, users of clustering implementations (toolkits) naturally assume that implementations of a deterministic clustering algorithm A have deterministic behavior, that is: (1) two different implementations I_1 and I_2 of A are interchangeable, producing the same clustering on a given input D, and (2) an implementation produces the same clustering solution when run repeatedly on D. We challenge these assumptions. Specifically, we analyze clustering behavior on 528 datasets, three deterministic algorithms (Affinity Propagation, DBSCAN, Hierarchical Agglomerative Clustering) and the deterministic portion of a fourth (K-means), as implemented in various toolkits; in total, we examined 13 algorithm-toolkit combinations. We found that different implementations of deterministic clustering algorithms make different choices, e.g., default parameter settings, noise insertion, input dataset characteristics. As a result, clustering solutions for a fixed algorithm-dataset combination can differ across runs (nondeterminism) and across toolkits (inconsistency). We expose several root causes of such behavior. We show that remedying these root causes improves determinism, increases consistency, and can even improve efficiency. Our approach and findings can benefit developers, testers, and users of clustering algorithms.**

***Index Terms*—ML testing, ML reliability, cluster analysis**

I. Introduction

The objective of cluster analysis (clustering) is to partition a given n-point dataset D into K "clusters"; points within a cluster are related or share a certain characteristic. A deterministic[1] clustering algorithm is designed to produce the same clustering solution C_K when run repeatedly on the same dataset D. Clustering toolkits are widely used in research and practice, including in critical sectors such as medicine, criminal justice, robotics, or finance [3]. Users who run such toolkits should be able to assume that the toolkits are reliable and interchangeable. Those affected by decisions made with toolkits' support should be able to assume that the toolkits are reliable. However, several factors undermine these assumptions. First, as the testings of implementations of ML algorithms is slight different from classical implementations, faults are not always due to bugs in the code, but also due to unaware of implementation subtleties or parameter combinations. Critical sectors might lack such clustering experts[2] to guarantee deterministic, consistent behavior. Second, our findings show that even after carefully tuning parameters – as experts would do – clustering toolkits are neither deterministic nor interchangeable: clustering solutions can vary across repeated runs due to implementation-induced nondeterminism, or across toolkits, due to inconsistent implementations of the same algorithm in different toolkits. Third, for unlabeled datasets, users or researchers simply have no reference point to assess an implementation's validity.

More precisely, we believe that end-users of deterministic clustering algorithms should be able to make two assumptions:

1. Consistency: Clustering toolkits consistently implement a certain algorithm, e.g., running algorithm A, as implemented in two different toolkits I_1 and I_2, on the same dataset D, yields the same clustering.

2. Determinism: The implementation of a deterministic algorithm is valid, i.e., computes the same clustering solution C when run repeatedly on the same dataset D.

We define consistency and determinism in Section II-A; check whether these properties hold in practice; find root causes when they do not hold; and find effective controls to re-establish these properties to a large extent.

We illustrate the consequences of nondeterminism and inconsistency in Table I on dataset "Breast Cancer Wisconsin (Original)"[3] [5]–[7]. The dataset has 699 instances and two classes: *benign* and *malignant*. Ideally, two different runs or two different toolkits would agree 100% when clustering this dataset, but we found that that was not the case. First, we observed nondeterminism (second column). When running R's implementation of the deterministic algorithm Affinity Propagation, two different runs can disagree on as many as 43 instances (while agreeing on the other 656). Next, we observed inconsistency (columns 3–5). For example, when running Affinity Propagation's implementations in Scikit-learn and in R, the two toolkits disagree on 36 points. More severely, for algorithm DBSCAN, MLpack's clustering solution disagrees with the solution obtained via Scikit-learn/R/Matlab

[1]We use the classical definition of *program determinism*: "a given input is always expected to produce the same output" [1], [2]; this is not to be confused with narrower definitions, e.g., deterministic thread scheduling.

[2]Forty-eight percent of global businesses, including healthcare, are suffering from a big data analytics skills gap [4].

[3]This breast cancer database was obtained from the University of Wisconsin Hospitals, Madison from Dr. William H. Wolberg. Available at https://www.openml.org/d/15

TABLE I
NONDETERMINISM AND INCONSISTENCY LEAD TO DISAGREEMENTS WHEN CLUSTERING THE 699-INSTANCE DATASET "BREAST CANCER WISCONSIN".

		R vs. itself	Scikit-learn vs. R	Scikit-learn/R/Matlab vs. MLpack	Scikit-learn/R vs. Matlab
	Algorithm	Affinity Propagation	Affinity Propagation	DBSCAN	Hierarchical Agglomerative
	Parameters	damping=0.916	damping=0.916	eps=0.5, minPts=5	linkage=Ward
	Issue	**Nondeterminism**	**Inconsistency**	**Inconsistency**	**Inconsistency**
#instances	*Disagree*	43	36	212	40
(out of 699)	*Agree*	656	663	487	659

on 212 points. Finally, for Hierarchical Clustering, Scikit-learn/R disagree with Matlab on 40 points. *Note how toolkits, parameters, or even individual runs, impact the outcome of a supposedly deterministic algorithm; this is problematic in general (undermining basic repeatability) or in high-stakes scenarios, e.g., when clustering is used in medical diagnosis*.

We now describe the experimental setup. We analyze clustering behavior on 528 datasets in several toolkits (Scikit-learn, R, MLpack, Matlab, TensorFlow); because the four algorithms are not available in all toolkits, in total we examined 13 algorithm-toolkit combinations.

Of the 528 datasets, about 400 were medical datasets, and the rest were benchmarking datasets. We examine medical datasets for two reasons. First, in healthcare, accuracy and correctness are paramount. Second, healthcare is a challenging domain, characterized by limited data volume and requiring model interpretability [8], which makes clustering via the algorithms we examine here more appropriate/preferred (as opposed to, say, Neural Networks; see Section VII for details).

We show that *default parameter settings* (representing latent assumptions developers make about how the toolkits will run) undermine both the determinism and interchangeability assumptions. For example, 4 out of the top-5 highest inconsistencies observed when running default implementations of Hierarchical Agglomerative Clustering were on Gene Expression for Oncology datasets (Section V-B).

In Section II we define deterministic clustering and discuss the experimental setup. Next, we quantify implementation-induced nondeterminism and inconsistency, find their root causes, and show how they can be alleviated. We do so for three deterministic algorithms – Affinity Propagation (Section III), DBSCAN (Section IV), Hierarchical Agglomerative Clustering (Section V) – and the deterministic part of K-means (Section VI). The end of each section presents actionable findings. In Section VII we review related work.

II. DEFINITIONS AND EXPERIMENTAL SETUP

We now define the main concepts and describe the setup for our approach.

A. Clustering, Determinism, and Consistency

Clustering is defined as follows. Given a set D of n points (d-dimensional vectors in the $\mathcal{R}^d$ space), the objective of clustering is to partition D into K non-overlapping subsets (clusters) $C_K = \{D_1, \ldots, D_i, \ldots, D_K\}$ such that intra-cluster distance between points (that is, within individual D_i's) is minimized. A *deterministic* clustering of D yields clustering solutions C'_K that are isomorphic to C_K.

TABLE II
STATISTICS ON DATASETS.

	Min	*Max*	*Geometric Mean*
Instances	27	9,989	454
Features (attributes)	2	61,360	39
K (# of clusters)	2	108	2.6

Determinism. A clustering implementation is deterministic if two different runs R and R' of the implementation on the same dataset D yield isomorphic clusterings C_R and $C_{R'}$.

Consistency. Two clustering implementations I_1 and I_2 are consistent if they yield isomorphic clusterings C_1 and C_2 when run on the same dataset D.

Violations of the aforementioned isomorphism requirements are due to nondeterministic or inconsistent implementations.

B. Datasets

We used 528 datasets from OpenML [9]. About 400 of these datasets are from medicine/bioinformatics, e.g., separating benign from malignant tumors, while the rest come from the Penn ML Benchmark [10], a benchmark suite specifically designed to evaluate ML implementations. Table II summarizes the characteristics of our datasets: on average, datasets have 454 instances, 39 dimensions, and 2.6 clusters.

C. Measuring Clustering Similarity and Accuracy

There are a myriad metrics for comparing two clusterings (partitionings) C and C' of an underlying set D. We use the *adjusted Rand index* (ARI) [11] as it is versatile, robust, intuitive, and therefore widely-used [12]. We have:

$$-1 \leq ARI(C, C') \leq 1$$

with $ARI(C, C') = 1$ indicating a perfect match (i.e., the same partitioning, or same solution); $ARI(C, C') = 0$ corresponds to random/independent clustering (i.e., as if C and C' elements were assigned to clusters randomly); and negative values of $ARI(C, C')$ indicate strong disagreement (i.e., few elements, if any, are in the same clusters in both C and C'). We use two bases for computing the ARI.

Accuracy is a term we use when comparing a clustering solution to Ground Truth (our datasets come with Ground Truth). When evaluating a clustering algorithm in the presence of Ground Truth, this is the main, "customer facing," external validation measure.

Consistency or *Mutual ARI* are terms we use when comparing two clustering solutions, C_1 and C_2, produced via different runs or different toolkits on the same dataset; mutual ARI is effective at internal validation, e.g., revealing inconsistency or nondeterminism without requiring Ground Truth.

```
procedure AffinityPropagation(D, K):

/* initialize K exemplars (cluster centers) */
C = {{d_c1},{d_c2},...,{d_cK}};

i = 0;
c = 0;
do {
  // refine clusters into C^new
  C^new = refine (C);
  if (d(C, C^new) < ε)
    c++; // no substantial changes in last c iterations
  else
    c = 0; // substantial changes

  C = C^new; // update solution
  i++;
}
while ((c < CONV_ITER) && (i < MAX_ITER));

return list of clusters C;
```

Fig. 1. Affinity Propagation pseudocode.

Accuracy and consistency are not interchangeable. For example, when clustering a dataset D, two toolkits can have ARI vs. Ground Truth $G_1 = 0$ and $G_2 = 0$ but their mutual ARI can be $M_{12} = 1$. That is, both toolkits yield the same bad solution. However, if $G_1 = 1$ and $G_2 = 1$ (perfect clustering) then necessarily $M_{12} = 1$ (perfect clustering is unique).

D. Algorithms and Toolkits

We studied 3 deterministic algorithms (Affinity Propagation, DBSCAN, Hierarchical Agglomerative Clustering) and the deterministic part of K-means. Each algorithm is described at the beginning of its respective section. We examined several toolkits: Matlab [13], MLpack [14], R [15], Scikit-learn [16], and TensorFlow [17]. The toolkits are popular: some have millions of users [13], [18], some are being used by large companies [19], [20].

III. Affinity Propagation

Affinity Propagation (AP) forms clusters by identifying "exemplars", i.e., one representative per cluster; initially all points are considered potential exemplars, and affinity (belonging) to a certain cluster is constructed iteratively via message-passing; the algorithm uses a damping factor – typically in the interval [0.5, 1) – to avoid moving points back-and-forth between clusters. We studied this algorithm's implementation in two toolkits: Scikit-learn and R.

A. Algorithm Overview

AP proceeds in two phases: initialization followed by iteration. Figure 1 shows the algorithm's pseudocode. AP is convergence-based, that is, it iterates until a convergence metric indicates the clusters are stable, or an iteration limit has been reached. Since these conditions (or parameters) are implementation-specific, nondeterminism can ensue.

During initialization, the algorithm deterministically picks the K initial points that are cluster representatives (exemplars). In the iteration phase, the current clustering solution C is refined into C^{new} at each iteration. If the distance between the current and previous iteration's solution $d(C, C^{new})$ is lower than a predefined threshold ε, the algorithm might have reached convergence (tracked by c). The iteration phase ends when either the clusters are not changing anymore (c >= CONV_ITER) or when a predefined total iteration limit (i >= MAX_ITER) has been reached. Examining the source code of different implementations for the same algorithm reveals that different toolkits use different default values for CONV_ITER and MAX_ITER, which can lead to inconsistency.

TABLE III
Bottom-5 and Mean Consistencies for Affinity Propagation; lower ARI values mean **stronger disagreement.**

	ARI: Scikit-learn vs. R	
Default	analcatdata_uktrainacc	0
	parity5	0
	sleuth_case1102	0
	rabe_166	0
	sleuth_ex1221	0
mean (all 528 datasets)		0.68
Forcing R to match Scikit-learn's #iterations	parity5	0.02
	mux6	0.11
	car-evaluation	0.12
	xd6	0.12
	threeOf9	0.14
mean (all 528 datasets)		0.95
Forcing Scikit-learn to match R's #iterations	dbworld-subjects	0
	schlvote	0
	hutsoff99_child_witness	0
	AP_Prostate_lung	0
	diggle_table_a1	0
mean (all 528 datasets)		0.94
Adaptive MAX_ITER	parity5	0
	sleuth_case1102	0
	rabe_166	0
	visualizing_slope	0
	analcatdata_vehicle	0
mean (all 528 datasets)		0.81

B. Inconsistency

We measure inconsistency using the mutual ARI (Section II-C). Ideally, the mutual ARIs would be 1 for all datasets, indicating that Scikit-learn and R yield the same solution. However, we found that toolkits disagree on 196 datasets. The 'Default' rows in Table III show the bottom-5 consistencies, i.e., the strongest disagreements. For example on parity5, the toolkits produce such different clustering solutions that they are practically unrelated: $ARI = 0.02$. The mean consistency is $ARI = 0.68$, well short of $ARI = 1$. The remainder of this section delves into inconsistency root causes and shows how addressing these root causes is effective at reducing inconsistency (the remaining Table III rows, explained in Sections III-E and III-F).

C. Case Study 1: Bounding the Number of Iterations

Different clustering implementations make different latent assumptions about convergence conditions, materialized in different default parameters.

We illustrate this in Figure 2 on Scikit-learn vs. R. By default, Scikit-learn bounds the total number of iterations MAX_ITER to 200, while R bounds it to 1000. The figure

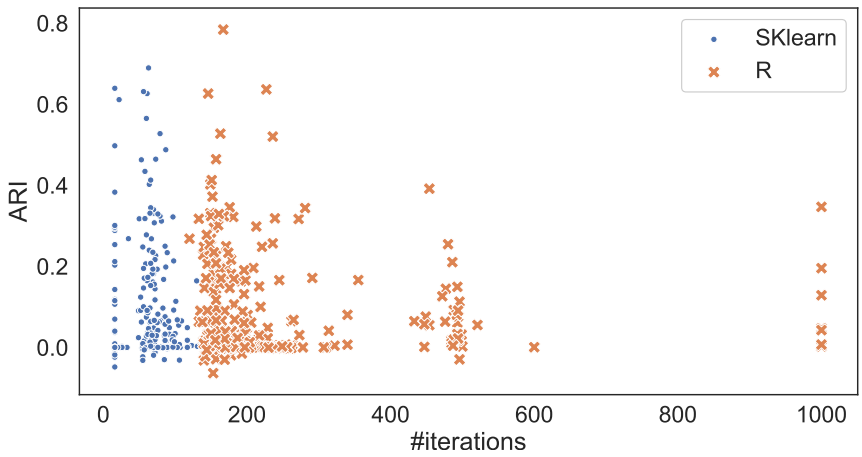

Fig. 2. Affinity Propagation's accuracy vs. #iterations in Scikit-learn and R.

shows the number of iterations (x-axis) required to cluster each dataset and the accuracy, i.e., ARI vs. Ground Truth (y-axis). Note how Scikit-learn takes substantially fewer iterations to cluster the datasets, yet without sacrificing precision.

A paired test on mean accuracy, that is, Scikit-learn's accuracy distribution vs. R's accuracy distribution, has shown no significant difference ($p-\text{value} > 0.1$). However, a paired test on #iterations until stopping (i in Figure 1) shows significant differences ($p-\text{value} \approx 0$): Scikit-learn's mean was 66 iterations, while R stops at 220 iterations, on average.

In fact, Scikit-learn always (for all datasets) terminates in fewer iterations compared to R. Regarding accuracy, we found that, out of 528 datasets: Scikit-learn has higher ARI than R for 232 of them; lower ARI for 200 of them; and the same ARI for 96 of them. To summarize, Scikit-learn is in a win-win, higher effectiveness-higher efficiency situation in 232 cases (fewer iterations, higher ARI).

Finally, note the "hard" limits for MAX_ITER at 200 and 1000, respectively – the 1000 vertical line is clearly visible for R in Figure 2 – if the implementation has not converged by then, the toolkit terminates. These parameters are up to the developers but their default values end up having substantial impact on accuracy, as shown next.

D. Under-iterating and Over-iterating

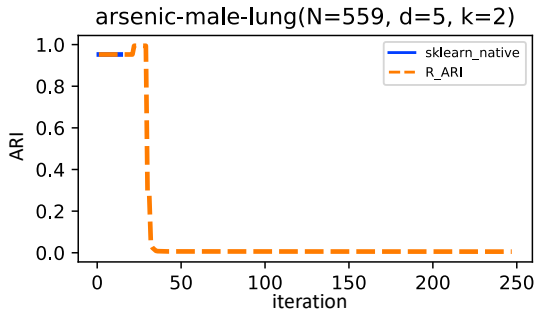

Fig. 3. Lose-lose due to over-iterating in R (orange dashed line); note the higher #iterations and lower final accuracy compared to Scikit-learn (blue).

Figure 3 shows the danger of over-iterating. The dataset is arsenic-male-lung; dataset characteristics are shown on top of the chart. Note how Scikit-learn exits after 16 iterations, at ARI=0.95, whereas R continues. Eventually R terminates after 231 iterations at ARI=0: a lose-lose scenario.

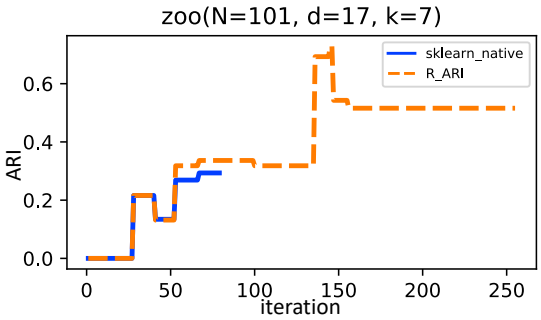

Fig. 4. Under-iterating – premature termination – leads to lower accuracy in Scikit-learn (blue solid line) compared to R (orange dashed line).

Conversely, Figure 4 shows the danger of under-iterating (on the zoo dataset). Note how Scikit-learn exits prematurely (blue solid line) after just 81 iterations, at ARI=0.29, whereas R (orange dashed line) continues; eventually R terminates, at ARI=0.52, after 174 iterations.

Table IV shows the highest margins for Scikit-learn and R, respectively. The first column contains the dataset name, the next four columns show the iterations and accuracy for Scikit-learn and R, respectively, while the last column shows the accuracy difference (absolute value).

Note how, on the arsenic-* datasets,[4] R's accuracy is essentially 0, whereas Scikit-learn's is 0.64–0.95. Moreover, Scikit-learn achieves this accuracy in just 16 iterations; this is due to Scikit-learn default setting CONV_ITER=15. The second half of the table shows those datasets where R has the upper hand, but we found the accuracy difference to be less than 0.22.

[4]Predicting the risk of certain cancers based on exposure to arsenic.

TABLE IV
HIGHEST ACCURACY MARGINS FOR AFFINITY PROPAGATION.

	Dataset	Scikit-learn		R		Accuracy
		Iterations	Accuracy	Iterations	Accuracy	gap
Scikit-learn's highest margin	arsenic-male-lung	16	0.95	247	0	0.95
	arsenic-female-lung	16	0.75	168	0	0.75
	arsenic-male-bladder	16	0.64	247	0	0.63
	kc1-top5	16	0.38	1000	0.04	0.34
	rabe_148	19	0.57	148	0.27	0.30
R's highest margin	zoo	81	0.29	255	0.52	0.22
	robot-failures-lp1	16	-0.05	454	0.13	0.18
	tokyo1	16	0	164	0.17	0.17
	AP_Omentum_Prostate	16	0	245	0.16	0.16
	AP_Prostate_Lung	16	0.04	206	0.20	0.16

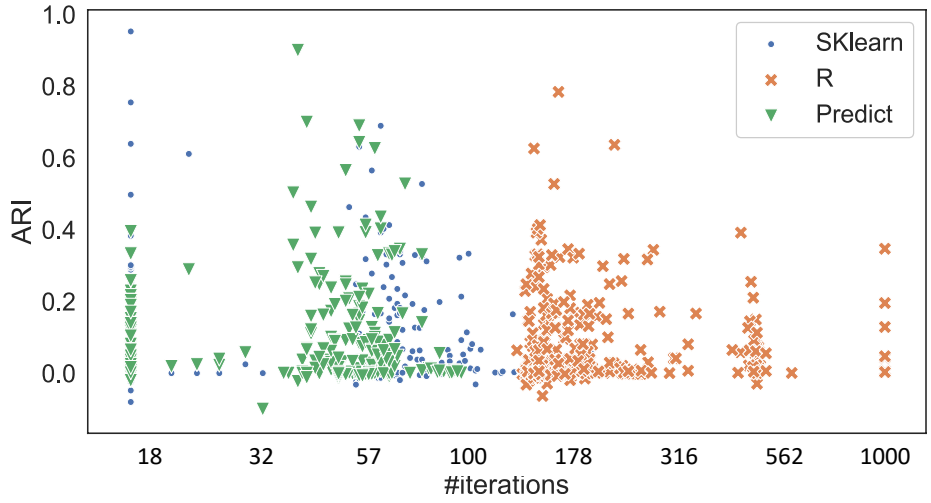

Fig. 5. ARI vs. #iterations: Scikit-learn predicted (green crosses), Scikit-learn default (blue triangles), R default (orange circles); for legibility, x-axis is logarithmic.

TABLE V
TOP-5 ACCURACY GAPS AFTER CONTROLLING FOR #ITERATIONS.

Dataset	Scikit-learn	R	Gap
schlvote	0.10	-0.06	0.17
ar3	0.12	0.23	0.10
dbworld-subjects	0.14	0.07	0.07
tecator	0.19	0.14	0.05
analcatdata_jap.	0.07	0.11	0.04

E. Heuristic 1: Consistent MAX_ITER

One potential solution for eliminating cross-toolkit inconsistencies would be to use the same MAX_ITER in both toolkits. Therefore, we ran experiments where, after obtaining R's terminating i (number of iterations), we forced Scikit-learn's to use it: MAX_ITER=i. After implementing this control into Scikit-learn, we were able to make two observations.

First, we noticed a slight decrease in Scikit-learn's accuracy, but the decrease was not statistically significant ($p-\text{value} = 0.16$). Second, the high-margin discrepancies between the two toolkits were removed or reduced substantially. In Table V we show the largest accuracy gaps after implementing this control. Note that accuracy differences were at most 0.17 (in stark contrast with Table IV where accuracy gaps were as high as 0.95). This demonstrates that forcing Scikit-learn to iterate longer yields no statistically significant gains in accuracy.

Finally, we measure how much consistency improves when forcing one toolkit to use the other's #iterations. The 'Forcing...' rows in Table III show consistency improving from 0.68 (default) to 0.94 or 0.95, respectively, which indicate this is an effective control.

F. Heuristic 2: Using an Adaptive MAX_ITER

An alternative solution to this problem (a fixed MAX_ITER does not fit all datasets) would be to use an "adaptive," per-dataset MAX_ITER. This showed promise as we were able to correlate $log(N)$ with i, the number of iterations at which the algorithm has terminated. Specifically, we ran an Ordinary Least Squares (OLS) regression where the dependent variable was the final number of iterations i, and the independent variable was $log(N)$; note that N is the number of points (instances) in the dataset. For Scikit-learn we found a good fit: $R^2 = 0.883$, $t-\text{value} = 42$, $p-\text{value} \approx 0$. For R, the regression did not find a good fit (Section III-G explains why).

Therefore we constructed a model where MAX_ITER was predicted by $log(N)$. Figure 5 shows how "tailoring" the termination to the dataset by replacing a fixed MAX_ITER with a predicted one effectively shifts all the Scikit-learn points

Scikit-learn	R
random_state = np.random.RandomState(0) # Remove degeneracies S += ((np. finfo (np.**double**).eps * S + np.finfo(np.**double**).tiny * 100) * random_state.randn(n_samples, n_samples))	**if** (!nonoise) randomMat <− matrix(rnorm(N * N),N) s <− s + (.Machine$double.eps * s + . Machine$double.xmin * 100) * randomMat

Fig. 6. Noise insertion code.

to the left (terminate sooner): the green crosses towards the left are Scikit-learn-predicted, while the blue triangles are the Scikit-learn-default. Moreover, a paired test on ARI indicated no significant ARI reduction ($p-\text{value} = 0.27$); that is, no precision is lost. However the test shows a statistically significant reduction in #iterations, from 66 to 57. To conclude, this approach improves efficiency without sacrificing precision. The 'Adaptive MAX_ITER' rows in Table III show how this improves consistency from 0.68 (default) to 0.81.

We emphasize that the point of this "tailoring" is not to improve accuracy but to underscore that defaults can be too rigid. Consequently, accuracy, efficiency, or both can suffer.

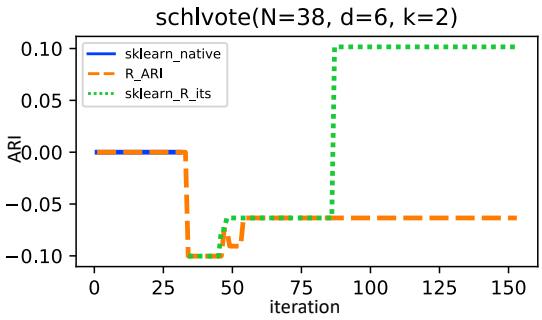

Fig. 7. Differences due to noise, after controlling for #iterations: by default, Scikit-learn would terminate quickly and at low accuracy (blue). Forcing Scikit-learn to keep iterating improves accuracy (green, dotted line). R's accuracy shown in orange dashed line.

G. Noise

Another source of inconsistency we discovered was the noise insertion policy. Essentially, toolkits choose to add "noise" to prevent degenerate clustering scenarios. Figure 6 shows the noise insertion code in Scikit-learn and R (noise insertion is ON by default in both toolkits). For both Scikit-learn and R, noise ranges from -1e-15*s to 1e-15*s(s is similarity matrix). However, R add random "noise", while Scikit-learn set fixed seed in the code, so the add "noise" is fixed. To quantify the impact of randomness of noise, we forced both toolkits to run for the same number of iterations, and compared the final outcomes, as discussed next.

1) Inconsistency: Figure 7 illustrates noise-induced inconsistency after controlling for #iterations (i.e., forcing Scikit-learn to "keep going" until it matches R's final number of iterations). Note how the difference in noise leads to a 0.2 gap in accuracy: 0.1 for Scikit-learn (green, dotted line) and -0.1 for R (orange, dashed line). After we turned off noise insertion, the two toolkits essentially achieve the same ARI ($p-\text{value} < 0.05$).

2) Nondeterminism: R inserts random noise, leading to nondeterminism, as discussed next (as expected, turning noise insertion off makes R's implementation deterministic).

TABLE VI
R: TOP-5 DIFFERENCES IN #ITERATIONS ACROSS RUNS.

Dataset	Iterations		
	Min	Max	Diff.
threeOf9	388	1000	612
scene	419	1000	581
corral	396	965	569
jungle	432	1000	568
parity5	437	1000	563

TABLE VII
R: TOP-5 DIFFERENCES IN ARI ACROSS RUNS.

Dataset	ARI		
	Min	Max	Gap
shuttle-landing-control	-0.07	0.44	0.51
Titanic	0.04	0.18	0.14
analcatdata_vehicle	0.01	0.10	0.09
parity5	-0.04	0.05	0.09
dbworld-subjects	0.06	0.15	0.09

When running R repeatedly on each dataset 30 times, out of 528 datasets, 107 had a nondeterministic number of iterations. In Table VI we show the top-5 such cases (minimum and maximum #iterations) sorted by the minimum-maximum difference. The numerous max. = 1000 values indicate that R failed to converge on that dataset for at least one run, and was force-stopped by the default MAX_ITER. We believe that this convergence nondeterminism – on the same dataset and with the same parameters – would surprise most R users.

Similarly, in Table VII we show the top-5 datasets, sorted by the minimum vs maximum accuracy gap, achieved when repeatedly running R, 30 times on the same dataset. We believe that understanding/avoiding such noise subtleties is well beyond the purview of a typical clustering user.

H. Actionable Findings

To conclude, our experiments have revealed that Affinity Propagation has deterministic behavior in Scikit-learn, and nondeterministic behavior in R due to flexible seed of noise insertion. Scikit-learn and R's implementations are mutually inconsistent due to default iterations and flexible seed of noise insertion.

These findings suggest that (a) users on R platform can track nondeterminism and inconsistency by turning off noise insertion. However, there is no parameter in Scikit-learn to turn off noise or set seed that makes the process to validate results and eliminate inconsistency harder. (b) Compared to R's performance, Scikit-learn is in a win-win scenario. we

suggest to use an adaptive MAX_ITER in Scikit-learn to improve determinism, consistency, and might even improve efficiency, i.e., high accuracy without over-iterating.

IV. DBSCAN

```
procedure DBSCAN(D, eps, minPts):

C = ∅;

foreach p in D {
    if (p not in a cluster) {
      N = set of points eps−reachable from p;
      if (|N| < minPts) {
        mark p as noise;
      }
      else {
        c = newCluster();
        add c to list of clusters C;
        foreach n in N {
          addToCluster(c,n);
          N' = set of points eps−reachable from N;
          N = N ∪ N';
        }
      }
    }
}

return list of clusters C;
```

Fig. 8. DBSCAN pseudocode.

DBSCAN forms clusters by looking for "dense" regions, i.e., regions with at least $minPoints$ separated by a maximum distance eps.[5] Unlike Affinity Propagation's variable #iterations, DBSCAN's number is fixed: in the general scenario we explore here, it practically executes $O(N^2)$ steps.

A. Algorithm Overview

Figure 8 shows the algorithm's pseudocode. For each unvisited point p, the algorithm "scans" its neighborhood (within eps distance). If there are at least $minPts$ in this neighborhood, p is a "core point" and will start a new cluster c; all of p's neighbors, and recursively their neighbors within eps, will be added to c. If, on the other hand, p does not have enough neighbors, it is declared noise and will not be part of any clusters.

B. Inconsistency

We studied this algorithm's implementation in four toolkits: Scikit-learn, R, MLpack, and Matlab. There was no variation across runs for any of the toolkits, so *our examined DBSCAN implementations were deterministic across runs*.

Therefore, we focus on inconsistency; specifically, we observed inconsistency when comparing MLpack with the other three toolkits.

[5]For DBSCAN, input order can introduce nondeterminism; we fixed input order to eliminate this potential issue.

TABLE VIII
ACCURACY FOR DBSCAN (ARI W.R.T. GROUND TRUTH): DEFAULT (TOP); CONTROLLED FOR *eps* (CENTER); HEURISTIC FOR *minPts* (BOTTOM).

Dataset	**Scikit-learn/ R/Matlab**	**MLpack**	**Max Gap**
Defaults	*eps=0.5, minPts=5*	*eps=1 minPts=5*	
zoo	-0.05	0.71	0.76
led7	0.33	0	0.33
ionosphere	-0.03	0.27	0.31
iris	0.75	1.00	0.25
acute-inflammations	0.20	0.44	0.24
mean (all 528 datasets)	0.006	0.009	
Controlled *eps*	*eps=0.5 minPts=5*	*eps=0.5 minPts=5*	
seismic-bumps	0.06	0.20	0.15
phoneme	0.13	-0.01	0.14
bank8FM	-0.03	0.02	0.06
seeds	0.06	0	0.05
acute-inflammations	0.20	0.23	0.03
mean (all 528 datasets)	0.006	0.006	
Heuristic *minPts*	*eps=0.5 minPts=d+1*	*eps=1 minPts=d+1*	
zoo	0	0.37	0.37
led7	0.33	0	0.33
smartphone-b.	0	0.31	0.31
qualitative-bankruptcy	0.19	0.48	0.28
acute-inflammations	0.17	0.44	0.27
mean (all 528 datasets)	0.006	0.012	

TABLE IX
BOTTOM-5 AND MEAN CONSISTENCIES FOR DBSCAN.

	MLpack vs Scikit-learn/R/Matlab	
Defaults	sonar	-0.015
	qual-bk.	-0.13
	vineyard	-0.10
	hayes-roth	-0.09
	pyrim	-0.09
mean (all 528 datasets)		0.79
Controlled for *eps*	seeds	-0.07
	bank8FM	0
	fri_c1_250_5	0
	fri_c1_500_5	0
	fri_c3_100_5	0
mean (all 528 datasets)		0.97
Heuristic *minPts*	hayes-roth	-0.09
	vineyard	-0.08
	qual-bk.	-0.078
	solar-flare	-0.073
	seeds	-0.059
mean (all 528 datasets)		0.81

a) Defaults: We started by running DBSCAN with defaults; we have default *minPts=5* for all three toolkits, default *eps=0.5* for Scikit-learn and R, and default *eps=1* for MLpack. We show the accuracy in the top third of Table VIII: the top-5 datasets, with the largest gaps between toolkits. The difference between MLpack and the other toolkits across all datasets was marginal, $p-\text{value} = 0.1$, albeit with a slightly higher mean (0.009 compared to 0.006). However, the difference could be quite large for specific datasets, e.g., for zoo, MLpack achieved ARI=0.71 while Scikit-learn and R's ARI=-0.05. The gap was noticeable for other datasets, too.

b) Controlling for eps: Next, we controlled for *eps* by setting MLpack's *eps* to 0.5. The results are shown in the middle of Table VIII: the gap was reduced considerably (at most 0.15 for dataset seismic-bumps). Actually, this control made the accuracies across all datasets statistically indistinguishable (three paired tests between the three toolkit combinations yielded $p-\text{value} \gg 0.1$; the mean was 0.006 for all toolkits). We have thus shown that, by controlling for **eps**, we can make MLpack's behavior more consistent with the other toolkits.

c) Using a heuristic for minPts: While by default *minPts=5* in all toolkits, R's DBSCAN package documentation mentions "Setting parameters for DBSCAN: minPts is often set to be dimensionality of the data plus one or higher" [21]. Therefore, we set *minPts=d+1*, where d is the dimensionality of the dataset; we present the results in the bottom rows of Table VIII. The difference between MLpack and the other toolkits across all datasets was significant, $p-\text{value} = 0.02$, and MLpack's mean in this scenario was the highest of all three scenarios: 0.012. The maximum gaps were also more prominent compared to the "controlled" version above (maximum gap was 0.37 for dataset zoo). Hence it appears that the heuristic is only effective for MLpack.

d) Mutual ARI: The measurements so far have used accuracy (ARI vs. Ground Truth). Table IX shows the mutual ARI results, before and after eliminating these root causes. We make several observations: with a default setting of *eps=1*, MLpack disagrees with the other toolkits substantially – note how the bottom-5 consistencies have *negative* ARI values, which signify disagreeing clustering solutions (worse than unrelated/random clustering which have ARI=0, see Section II-C). Across all datasets, we have mean ARI=0.79. The situation improves when controlling for *eps* (middle of the table, note that mean ARI=0.97). Finally, when using *eps=1* and *minPts=d+1*, MLpack again tends to disagree (mean ARI=0.81).

C. Actionable Findings

To conclude, our experiments have revealed that DBSCAN has deterministic behavior in Scikit-learn, R, Matlab and MLpack. MLpack's implementation can be inconsistent with the other toolkits due to its different default *eps*.

These findings suggest that MLpack can gain consistency and accuracy: using a common *eps* makes MLpack more consistent with the other toolkits, while using a heuristic *minPts* improves MLpack's accuracy.

V. Hierarchical Agglomerative Clustering

Hierarchical clustering (we use its agglomerative variant) proceeds bottoms-up by first considering each point a cluster and then iteratively merging clusters based on linkage criteria (minimizing distance between points, usually).

A. Algorithm Overview

Figure 9 shows the algorithm's pseudocode. Initially, each point d_i is its own cluster. Next, inter-cluster distances $\mathsf{d}(C_i,C_j)$ are computed and the closest clusters C_{mi},C_{mj} are merged.

```
procedure HierarchicalAgglomerativeClustering(D,k) :

// D = {d1,d2,...,dn}

// start with N singleton clusters
C = {{d1},{d2},...,{dn}};

do {
  mi,mj = argmin (d(Ci,Cj)) over all (i,j) pairs in C
  C' = merge(Cmi,Cmj);
  remove Cmi,Cmj from list of clusters C;
  add C' to list of clusters C;
}
while (|C| > k);

return list of clusters C;
```

Fig. 9. Hierarchical Agglomerative pseudocode.

The algorithm continues until it reaches the desired number of clusters k.

B. Inconsistency

We studied this algorithm's implementation in three toolkits: Scikit-learn, R, and MATLAB. Our experiments have revealed that Hierarchical clustering implementations are deterministic. Therefore, we only focus on inconsistency.

a) Accuracy: We found 173 cases where the toolkits' ARIs (accuracies) on the same dataset differ by more than 0.1. In the top half of Table X we show the top-5 datasets by accuracy gap between the three toolkits. Four out of these five datasets come from the Gene Expression for Oncology repository GEMLeR by Stiglic and Kokol [22]: AP_Prostate_Lung, AP_Omentum_Prostate, AP_Prostate_Kidney, and AP_Endometrium_Prostate.

We determined that one source of differences was the *linkage criterion* (distance function), which was different for each toolkit: *Ward* vs. *Complete* vs. *Single* for Scikit-learn, R, and MATLAB, respectively. Since *Ward* is uniformly supported in all three toolkits, we set the linkage to *Ward*, and report the results in the bottom half of Table X. Using the same linkage not only improves consistency, but also increases accuracy for both R and MATLAB.

We also found an implementation difference so substantial that it is impossible to control for by just changing input parameters: R's implementation is optimized for time via fast distance computation (Nearest-neighbor chain algorithm [23]). Per its authors [24], R is the only clustering toolkit to use this distance computation method. Extricating the distance computation code to force consistency with other toolkits would be a substantial endeavor (as it is pervasive throughout the implementation). We leave this endeavor to future work.

b) Mutual ARI: The mutual ARIs are presented in Table XI. For bottom-5 consistencies, note the negative values in default mode (top rows); the mean consistency across all datasets was 0.14–0.41, which is way lower than DBSCAN defaults (0.79–0.97) or Affinity Propagation defaults (0.95).

Controlling for linkage substantially improves consistency: while some datasets' mutual ARIs are around 0 (bottom rows of Table XI) the mean mutual ARI has increased to 0.93–0.94.

TABLE X
ACCURACY FOR HIERARCHICAL AGGLOMERATIVE CLUSTERING: DEFAULT (TOP); WITH SCIKIT-LEARN'S DEFAULT LINKAGE *Ward* (BOTTOM).

	Dataset	Scikit-learn	R	MATLAB	Max Gap
Defaults		*l=Ward*	*l=Complete*	*l=Single*	
	synthetic_control	-0.05	-0.05	1	1.05
	AP_Prostate_Lung	0.89	-0.00	-0.00	0.90
	AP_Omentum_Prostate	0.87	-0.00	-0.00	0.87
	AP_Prostate_Kidney	0.85	-0.00	-0.00	0.85
	AP_Endometrium_Prostate	0.85	0.00	0.00	0.85
	mean (all 528 datasets)	0.11	0.12	0.11	
l=Ward	socmob	0.17	0.50	0.17	0.33
	analcatdata_supreme	0.25	0.04	-0.06	0.31
	corral	0.30	0.30	0.03	0.26
	analcatdata_boxing2	0.01	0.19	0.02	0.17
	vinnie	0.27	0.30	0.17	0.45
	mean (all 528 datasets)	0.11	0.12	0.11	

TABLE XI
BOTTOM-5 AND MEAN CONSISTENCIES FOR HIERARCHICAL AGGLOMERATIVE CLUSTERING.

	Scikit-learn vs. R		Scikit-learn vs. Matlab		R vs. Matlab	
Defaults	mbagrade	-0.11	rabe_266	-0.06	mbagrade	-0.06
	molecular-biology-promoters	-0.11	diggle_table_a2	-0.05	rabe_266	-0.06
	planning-relax	-0.10	synthetic_control	-0.05	synthetic_control	-0.05
	tic-tac-toe	-0.10	lupus	-0.04	allbp	0.33
	hepatitisC	-0.08	analcatdata_uktrainacc	-0.04	parity5	-0.03
mean (all 528 datasets)		0.41		0.14		0.26
l=Ward	tic-tac-toe	-0.10	mux6	-0.01	optdigits	-0.08
	optdigits	-0.08	analcatdata_boxing2	-0.01	mbagrade	0.01
	mbagrade	-0.02	parity5_plus_5	0	mux6	0
	threeOf9	0	car	0	analcatdata_boxing2	0
	profb	0	threeOf9	0	car	0
mean (all 528 datasets)		0.94		0.93		0.93

C. Actionable Findings

To conclude, our experiments have revealed that Hierarchical Agglomerative Clustering has deterministic behavior in Scikit-learn, R, and Matlab. However, all three implementations are mutually inconsistent due to different default linkage; setting linkage to "Ward" is an effective consistency measure.

VI. KMEANS

K-means forms clusters by assigning points to their closest cluster center. Given initial "centroids," K-means assigns each point to the closest centroid, calculates the new centroids (means of updated clusters), and repeats the process until clusters are stable. While the choice of initial centroids is nondeterministic, the iteration phase is deterministic. Therefore, our strategy was to choose the same centroids for all implementations and study implementation-induced nondeterminism and inconsistency, due to the iteration phase. We studied K-means in four toolkits: Scikit-learn, R, Matlab, and Tensorflow.

A. Algorithm Overview

Figure 10 shows the algorithm's pseudocode. K-means has two phases: initialization and iteration. In the initialization phase, the algorithm is nondeterministic as it randomly picks K points as initial centroids $\{C_1, \ldots, C_K\}$. In the deterministic iteration phase, each data point is assigned to the closest center C_j and centers C_m are recomputed (as means of updated clusters). The iteration phase ends when the clusters are not changing anymore (C_m is not changing); or when the objective (sum of squared Euclidean distances of observations from their cluster centers) is minimized; or when a predefined maximum number of iterations is reached (iter $\geq$MAX_ITER). Therefore, for the same starting points, we would expect different implementations to converge to the same result. However, upon examining the source code, we

```
/*                    K−MEANS                    */
// Input: dataset S = x_1, ..., x_n; number of clusters K
// start with empty clusters
   S_1 = ∅; ...; S_K = ∅;
// INITIALIZATION PHASE: initialize centroids randomly
// {C_1, ..., C_K} = {x_{r_1}, ..., x_{r_K}}

iter = 0;
do {// ITERATION PHASE
    // Assignment step: add each point x_i to its closest centroid
    for (i=1; i <= n; i++) {
    // " tie " exists when there are at least one minimum
        m = argmin Σ_{j=1}^{K} ||x_i − C_j||
        S_m = S_m ∪ {x_i}
    }
    // Update step: recompute centroids to be the mean of the
         updated clusters
    for (j=1; j <= K; j++) {
        C_m = ( Σ_{l=1}^{|C_m|} x_lm)/|C_m|
    }
      iter ++;
}
while ((clusters still changing || objective > minObjective)
       && iter < MAX_ITER);
```

Fig. 10. K-means pseudocode.

have found that different toolkits make different choices (e.g., stopping conditions or tie-breaking) that introduce inconsistency. Specifically, K-means is NP-hard when seeking a global optimum, so toolkits employ heuristics which substantially improve efficiency, but risk converging to a local optimum.

B. Inconsistency

We show the progression toward stronger consistency, starting from default parameters and then applying stronger controls: Table XII shows the number of datasets that toolkits disagree on, while Table XIII shows the mean mutual ARIs and the strongest disagreements.

a) Defaults: The lowest consistencies are between R and other toolkits: ARIs as low as -0.06 for four datasets, and disagreements on 369–376 datasets. This is due to R's default implementation, including stopping conditions.[6]

b) Stop conditions: The most important consistency parameter is MAX_ITER. By default MAX_ITER=10 for R, Matlab uses MAX_ITER=100 and Scikit-learn uses MAX_ITER=300. Since we found that 96.9% of datasets finish in 40 iterations or less, we set MAX_ITER=100 for all toolkits. Table XII shows that R's disagreement with the other toolkits reduces substantially, from 369–370 disagreements to just 15–21; Table XIII shows that the mean mutual ARI increases from 0.75 to 0.99.

Scikit-learn uses a parameter "tolerance," i.e., the relative difference in objectives between two iterations, as one of the stop conditions. By default, TOL=0.0001 in Scikit-learn; its equivalent would be TOL=0 in the other toolkits. We found that setting Scikit-learn's TOL=0 yields a small increase in consistency; due to the small magnitude of this improvement (visible at the third decimal place) we omit it from Table XII.

c) "Tie-breaking" at first iteration: Even after the aforementioned controls, we still observe inconsistencies. For example, R and Scikit-learn have inconsistencies on 18 out of 497 datasets; Tensorflow and R have inconsistencies on 21 out of 497 datasets. Most of these inconsistencies are visible after the first iteration. When we compared label assignments between toolkits after the first iteration, we found that toolkits break "ties" (i.e., assign observations that have the same Euclidean distances to cluster centers) differently. For example, Scikit-learn assign points with ties to the cluster that has the higher index, that is quite arbitrary tie breaking; Matlab resolves ties by keeping last step's assignments – it will prefer not to move a point if it becomes tied. These tie breaking-induced inconsistencies persist after the first iteration, as later steps are deterministic. Therefore, our next control was to avoid starting points that have equal distance to points to be clustered. This measure increased inconsistencies for 25 datasets; bottom-5 consistencies are shown in the last 6 rows Table XIII. Note that mutual ARIs are at least 0.99.

Table XII shows 6–23 datasets that still have inconsistencies after controlling for first iteration tie-breaking. These inconsistencies are due to inherent ties in datasets (certain points are equidistant to cluster centers) and cannot be avoided by changing parameters or starting points.

C. Actionable Findings

To conclude, our experiments have revealed that Scikit-learn, R, Matlab and Tensorflow's K-means implementations are deterministic, but mutually inconsistent due to heuristics, stop conditions, and tie-breaking. These findings suggest that controlling for MAX_ITER and tie-breaking strategy are effective measures for achieving high consistency.

VII. Related Work

Testing, verification, and validation of Machine Learning implementations is an emerging area. Prior efforts have focused on supervised learning approaches, mostly Neural Networks (NNs)/Deep Learning [27], rather than unsupervised learning (e.g., clustering). While NNs are popular and successful, challenges such as limited training data, unlabeled data, or interpretability[7] make "traditional" clustering preferable.

Yin et al. [28] and Musco et al. [29] have used statistical tests to compare variation across runs and toolkits for clustering algorithms. They ran their approach on 7 algorithms (DBSCAN, Affinity Propagation, Hierarchical, K-means, K-means++, Gaussian mixtures, Spectral). Their goal was to characterize accuracy distributions via descriptive statistics, i.e., their efforts were observational: no attempt was made to explain what causes differences, or to control for parameters and observe the effect of these controls. In contrast, we look into cause and effect. First, what causes differences across runs or toolkits? Second, what changes (and how) after we control, e.g., for parameters. We use a substantially higher number of datasets (528 vs. their 162) which strengthens the relevance of our statistical findings.

Kriegel et al. [30] have also questioned the assumption that toolkits are interchangeable. They have compared (mostly) nondeterministic algorithms and a wide range of toolkits, but in terms of runtime efficiency. Our focus is different: root causes of toolkits' accuracy differences. They used a single "master" dataset of 500K Twitter locations, from which they sampled smaller datasets. We used a much broader, 528-dataset setup.

In the Machine Learning community, efforts have focused on improving clustering performance. For example Ben-Hur et al. [31] have experimented with varying the K used in hierarchical clustering to find a value that yields the highest stability (a condition stronger than determinism). To improve the performance of K-means, Fred [32] has proposed "voting K-means", that is, using a majority-vote strategy to form more accurate clusters by running k-means repeatedly and eliminating "dissenting" runs. They varied K, but experimented on two datasets only. Our focus is different – exposing toolkit-induced nondeterminism; our dataset range is much broader.

[6] R uses "Hartigan-Wong" heuristics [25] by default, whereas Scikit-learn and Matlab use "Lloyd" heuristics [26].

[7] In domains such as healthcare interpretability "is crucial for convincing the medical professionals about the actions recommended" [8].

TABLE XII
NUMBER OF DATASETS THAT HAVE INCONSISTENCIES FOR K-MEANS FOR EACH CONTROLLING STEP.

#Datasets	TF vs. R	TF vs. Scikit-learn	TF vs. Matlab	R vs. Scikit-learn	R vs. Matlab	Scikit-learn vs. Matlab
Default parameters	369	82	29	376	369	58
Fixed *ITER=100*	21	31	29	18	15	4
Control first-iteration tie	15	23	22	16	13	6

TABLE XIII
BOTTOM 5 CONSISTENCIES FOR K-MEANS FOR EACH CONTROLLING STEP.

		TF vs. R	TF vs. Sk	TF vs. Matlab	R vs. Sk	R vs. Matlab	Sk vs. Matlab
Default parameters	analcatdata_vehicle	-0.02	0.43	0.43	-0.02	-0.02	1
	analcatdata_chlamydia	-0.01	0.64	0.64	-0.01	-0.01	1
	rabe_266	-0.01	0.64	0.64	-0.01	-0.01	1
	AP_Breast_Kidney	0.00	0.76	0.76	0.00	0.00	1
	fri_c4_100_50	-0.09	1	1	-0.09	-0.09	1
mean		0.75	0.99	0.99	0.75	0.75	1
Fixed *ITER=100*	analcatdata_vehicle	1	0.43	0.43	0.43	0.43	1
	monks-problem3	1	0.46	1	0.46	1	0.46
	glass	0.59	0.59	0.59	1	1	1
	rabe_266	1	0.64	0.64	0.64	0.64	1
	analcatdata_boxing1	1	0.69	0.69	0.69	0.69	1
mean		0.99	0.98	0.99	0.99	0.99	1
Control first-iteration tie	solar-flare	0.31	0.58	0.58	0.43	0.43	1
	analcatdata_vehicle	1	0.43	0.43	0.43	0.43	1
	led7	0.41	0.83	0.82	0.40	0.42	0.90
	LED-display-domain-7digit	0.56	0. 73	0.95	0.56	0.54	0.69
	cleveland-nominal	0.52	0.92	0.92	0.52	0.52	1
mean		0.99	0.99	0.99	0.99	0.99	1

VIII. CONCLUSIONS

Testing of Machine Learning implementations is an emerging area. We focus on deterministic clustering algorithms. In theory, these algorithms should produce the same clustering solution across runs and toolkits. In practice, we discovered that this assumption breaks, leading to nondeterministic implementations and inconsistency across toolkits. For each algorithm, we were able to expose the diverse root causes of nondeterminism or inconsistency: default parameter settings, noise insertion, distance metrics, termination criteria. Controlling for these sources can eliminate nondeterminism and bring several different implementations of the same algorithm more in line with each other.

We believe that our approach and actionable findings have the potential to improve clustering reproducibility as well as reliability, which benefits a wide range of clustering users.

ACKNOWLEDGMENTS

We thank the anonymous reviewers for their valuable feedback. Research was sponsored by the Army Research Laboratory and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation here on.

REFERENCES

[1] R. L. Bocchino, Jr., V. S. Adve, S. V. Adve, and M. Snir, "Parallel programming must be deterministic by default," in *Proceedings of the First USENIX Conference on Hot Topics in Parallelism*, ser. HotPar'09. Berkeley, CA, USA: USENIX Association, 2009, pp. 4–4. [Online]. Available: http://dl.acm.org/citation.cfm?id=1855591.1855595

[2] C. Manning, P. Raghavan, and H. Schütze, "Introduction to information retrieval," *Natural Language Engineering*, vol. 16, no. 1, pp. 100–103, 2010.

[3] "Cluster analysis," May 2019. [Online]. Available: https://en.wikipedia.org/wiki/Cluster_analysis

[4] J. Bresnick, "48% of businesses, including healthcare, face big data skills gap," May 2017, https://healthitanalytics.com/news/48-of-businesses-including-healthcare-face-big-data-skills-gap.

[5] 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.

[6] "Breast cancer diagnosis and prognosis via linear programming," *Oper. Res.*, vol. 43, no. 4, pp. 570–577, Aug. 1995. [Online]. Available: http://dx.doi.org/10.1287/opre.43.4.570

[7] 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

[8] R. Miotto, F. Wang, S. Wang, X. Jiang, and J. T. Dudley, "Deep learning for healthcare: review, opportunities and challenges," *Briefings in Bioinformatics*, vol. 19, no. 6, pp. 1236–1246, 05 2017. [Online]. Available: https://doi.org/10.1093/bib/bbx044

[9] "OpenML," May 2019, https://www.openml.org/.

[10] R. S. Olson, W. La Cava, P. Orzechowski, R. J. Urbanowicz, and J. H. Moore, "Pmlb: a large benchmark suite for machine learning evaluation and comparison," *BioData Mining*, vol. 10, no. 1, p. 36, Dec 2017.

[11] L. Hubert and P. Arabie, "Comparing partitions," vol. 2, pp. 193–218, 02 1985.

[12] D. Steinley, "Properties of the hubert-arable adjusted rand index." *Psychological methods*, vol. 9, no. 3, p. 386, 2004.

[13] "Mathworks fast facts," May 2019, https://www.mathworks.com/company/aboutus.html.

[14] "mlpack: fast, flexible C++ machine learning library," May 2019, https://www.mlpack.org/.

[15] "The R Project for Statistical Computing," May 2019, https://www.r-project.org/.

[16] "scikit-learn: Machine Learning in Python," May 2019, https://scikit-learn.org/.

[17] "Tensorflow github," https://github.com/tensorflow/tensorflow.

[18] M. Hornick, "Oracle r technologies overview," https://www.oracle.com/assets/media/oraclertechnologies-2188877.pdf.
[19] "Companies using tensorflow," https://www.tensorflow.org/.
[20] "Who is using scikit-learn?" May 2019, http://scikit-learn.org/stable/testimonials/testimonials.html.
[21] M. Hahsler, "R's dbscan package v. 1.1.3," May 2019, https://cran.r-project.org/web/packages/dbscan/index.html.
[22] G. Stiglic and P. Kokol, "Stability of ranked gene lists in large microarray analysis studies," *Journal of biomedicine & biotechnology*, vol. 2010, p. 616358, 06 2010.
[23] "Nearest-neighbor chain algorithm," May 2019. [Online]. Available: https://en.wikipedia.org/wiki/Nearest-neighbor_chain_algorithm
[24] "The R Project: Hierarchical Clustering," May 2019, https://svn.r-project.org/R/trunk/src/library/stats/R/hclust.R.
[25] J. Hartigan and M. Wong, "Algorithm AS 136: A K-means clustering algorithm," *Applied Statistics*, pp. 100–108, 1979.
[26] S. Lloyd, "Least squares quantization in pcm," *IEEE Trans. Inf. Theor.*, vol. 28, no. 2, pp. 129–137, Sep. 2006. [Online]. Available: http://dx.doi.org/10.1109/TIT.1982.1056489
[27] T. Gehr, M. Mirman, D. Drachsler-Cohen, P. Tsankov, S. Chaudhuri, and M. Vechev, "Ai2: Safety and robustness certification of neural networks with abstract interpretation," in *2018 IEEE Symposium on Security and Privacy (SP)*, May 2018, pp. 3–18.
[28] X. Yin, V. Musco, I. Neamtiu, and U. Roshan, "Statistically rigorous testing of clustering implementations," in *The First IEEE International Conference on Artificial Intelligence Testing, AITEST, April 2019*, April 2019.
[29] V. Musco, X. Yin, and I. Neamtiu, "Smokeout: An approach for testing clustering implementations," in *12th IEEE International Conference on Software Testing, Verification and Validation, ICST 2019*, April 2019.
[30] H.-P. Kriegel, E. Schubert, and A. Zimek, "The (black) art of runtime evaluation: Are we comparing algorithms or implementations?" *Knowl. Inf. Syst.*, vol. 52, no. 2, pp. 341–378, Aug. 2017.
[31] A. Ben-Hur, A. Elisseeff, and I. Guyon, "A stability based method for discovering structure in clustered data," in *Proceedings of the 7th Pacific Symposium on Biocomputing, PSB 2002, Lihue, Hawaii, USA, January 3-7, 2002*, 2002, pp. 6–17. [Online]. Available: http://psb.stanford.edu/psb-online/proceedings/psb02/benhur.pdf
[32] A. Fred, "Finding consistent clusters in data partitions," in *In Proc. 3d Int. Workshop on Multiple Classifier*. Springer, 2001, pp. 309–318.