ABSTRACT

EXPOSING AND FIXING CAUSES OF INCONSISTENCY AND NONDETERMINISM IN CLUSTERING IMPLEMENTATIONS

by
Xin Yin

Cluster analysis aka Clustering is used in myriad applications, including high-stakes domains, by millions of users. Clustering users should be able to assume that clustering implementations are correct, reliable, and for a given algorithm, interchangeable. Based on observations in a wide-range of real-world clustering implementations, this dissertation challenges the aforementioned assumptions.

This dissertation introduces an approach named SmokeOut that uses differential clustering to show that clustering implementations suffer from nondeterminism and inconsistency: on a given input dataset and using a given clustering algorithm, clustering outcomes and accuracy vary widely between (1) successive runs of the same toolkit, i.e., nondeterminism, and (2) different toolkits, i.e, inconsistency. Using a statistical approach, this dissertation quantifies and exposes statistically significant differences across runs and toolkits. This dissertation exposes the diverse root causes of nondeterminism or inconsistency, such as default parameter settings, noise insertion, distance metrics, termination criteria. Based on these findings, this dissertation introduces an automatic approach for locating the root causes of nondeterminism and inconsistency.

This dissertation makes several contributions: (1) quantifying clustering outcomes across different algorithms, toolkits, and multiple runs; (2) using a statistical rigorous approach for testing clustering implementations; (3) exposing root causes of nondeterminism and inconsistency; and (4) automatically finding nondeterminism and inconsistency's root causes.

EXPOSING AND FIXING CAUSES OF INCONSISTENCY AND NONDETERMINISM IN CLUSTERING IMPLEMENTATIONS

by
Xin Yin

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

December 2020

APPROVAL PAGE

EXPOSING AND FIXING CAUSES OF INCONSISTENCY AND NONDETERMINISM IN CLUSTERING IMPLEMENTATIONS

Xin Yin

Dr. Iulian Neamtiu, Dissertation Advisor — Date
Professor, NJIT

Dr. Ali Mili, Committee Member — Date
Professor and Associate Dean for Academic Affairs, NJIT

Dr. Usman Roshan, Committee Member — Date
Associate Professor of Computer Science, NJIT

Dr. Ioannis Koutis, Committee Member — Date
Associate Professor of Computer Science, NJIT

Dr. Ji Meng Loh, Committee Member — Date
Associate Professor of Mathematical Sciences, NJIT

BIOGRAPHICAL SKETCH

Author:	Xin Yin
Degree:	Doctor of Philosophy
Date:	December 2020

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science
  New Jersey Institute of Technology, Newark, New Jersey 2020
- Master of Science, Statistics
  University of Connecticut, Storrs, Connecticut, 2014
- Bachelor of Science, Mathematics and Applied Mathematics
  Zhejiang gongshang University, Hangzhou, Zhejiang, China, 2012

Major: Computer Science

Presentations and Publications:

Xin Yin, Iulian Neamtiu, "Automatic Detection of the Root Causes of Clustering Nondeterminism and Inconsistency," *the 14th International Conference on Software Testing, Verification, and Validation (ICST)*, in preparation, 2021.

Sydur Rahaman, Iulian Neamtiu, Xin Yin, "Categorical Information Flow Analysis for Understanding Android Identifier Leaks," *the 28th Network and Distributed System Security Symposium (NDSS)*, submitted, February 2021.

Xin Yin, Iulian Neamtiu, Saketan Patil, Sean T Andrews, "Implementation-induced Inconsistency and Nondeterminism in Deterministic Clustering Algorithms," *the 13th International Conference on Software Testing, Verification, and Validation (ICST)*, 2020.

Vincenzo Musco, Xin Yin, Iulian Neamtiu, "SmokeOut: An Approach for Testing Clustering Implementations," *the 12th International Conference on Software Testing, Verification, and Validation (ICST)*, Tool Track, 2019.

Xin Yin Vincenzo Musco, Iulian Neamtiu, Usman Roshan, "Statistically Rigorous Testing of Clustering Implementation," *The First IEEE International Conference on Artificial Intelligence Testing*, 2019.

Xin Yin, Vincenzo Musco, Iulian Neamtiu, “A Study on the Fragility of Clustering,” *IBM AI Systems Day 2018*, MIT-IBM Watson AI Lab, MIT, 2018.

Jia He, Changying Du, Fuzhen Zhuang, Xin Yin, Qing He, Guoping Long, “Online Bayesian max-margin subspace multi-view learning,” *the 25th International Joint Conference on Artificial Intelligence (IJCAI)*, 2016.

Dedicated to my family

ACKNOWLEDGMENT

First and foremost, I would like to express my sincere gratitude to my advisor Iulian Neamtiu for the continuous support of my Ph.D study and related research, for his patience, motivation, and immense knowledge. Without his research vision and utmost patience with my at times floundering process, this thesis would not be possible. Doing my current research was one of the best decisions that I have ever made. Iulian always pushed me towards interesting and important questions that solved real world problems.

This dissertation would also not have been possible without Dr. Vincenzo Musco. He brought me to research in Software Engineering and taught me lessons. He enlightened me at the first glance of research and I learned a lot of essential skills with his help. Thanks for his patience for guiding my research from time to time.

My gratitude also goes to my awesome thesis committee members: Ali Mili, Usman Roshan, Ioannis Koutis and Ji Meng Loh, for their insightful comments and encouragement, but also for the valuable suggestions to improve my research quality. I had the wonderful opportunity to collaborate with Usman Roshan and he broadened our research on the machine learning community.

I would like to thank the computer science department for offering the opportunity to be the teaching assistant. I enjoyer tutoring students and helping them build confidence in their ability to achieve. I also want to thank the National Science Foundation to get financial support and work as a research assistant in multiple projects and gain a lot of experience.

I thank my collaborators for the stimulating discussions and supportive suggestions. Sean Andrew and Patil Saketan have been amazing collaborators and this work would not have been possible without them. I am also very grateful to

Sydur Rahaman for his support to participate in the project to investigate android identifier leaks that enrich my horizons.

A very special thanks to the Mobile Profiler team I worked on at Facebook during this summer. Thanks for my mentor Nathan Slingerland at Facebook for his fitness and flexible training plan and timely and effective feedback on my work that eventually made my internship succeed. Thanks to all my peers Riham Selim, Cheng Chang, Anshuman Chadha frankly and fairly reviews on my work, that pulls me back on the right track quickly.

Last, but not the least, I would like to thank my family: my parents always believing in me and encouraging me to achieve my dreams. And my grandparents, my aunts and uncles, my cousin sister, they took care of me when I was sick in hospital in my fourth year. Special thanks to my doctor, for in time treatment, so I can go back and continue my study. They saved me and supported me to complete my PhD.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

Table **Page**

LIST OF TABLES
(Continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Introduction

Cluster analysis (*Clustering*) is an unsupervised learning technique used to group together entities that are related or share similar characteristics. While clustering is a well-established area with research going back to the 1950s, there is a pressing need for approaches to testing clustering implementations due to several converging factors.

First, supervised and unsupervised learning have started to permeate software products, from "smart" home devices [15] to self-driving platforms [42] and predictive analytics [45]. These implementations make critical decisions themselves or are used to aid decision-making (e.g., autonomous driving or financial fraud detection).

Second, there has been a proliferation of clustering implementations, mostly in the form of software toolkits (e.g., MATLAB and R each offer more than 100 clustering packages [6, 7]). These implementations are run by millions of users [9, 33] including non ML-experts (from life scientists to medical professionals) who should be able to assume that the implementations are correct.

Third, software engineers are under pressure to incorporate/adopt Machine Learning into software products and processes [18, 21, 35]; engineers should be able to (reasonably) assume that clustering implementations are reliable and interchangeable, i.e., for a given algorithm, its implementation is correct and has no negative impact on the clustering outcome.

In Table 1.1 we illustrate these issues by highlighting nondeterminism and inconsistency in widely used clustering implementations when run on critical datasets: security, safety, public health, etc. This table shows two major issues. First, clustering

Table 1.1 Nondeterminism and Inconsistency Issues in Clustering Implementations

Issue	**Nondeterminism**	**Inconsistency**	**Inconsistency**	**Inconsistency**
Clustering Algorithm	Affinity Prop.	Affinity Prop.	DBSCAN	Hierarchical
Implementation	R	Scikit-learn vs R	Scikit-learn/R/Matlab vs MLpack	Scikit-learn/R vs Matlab
Dataset				
Coal mine seismic risk			●	
Pest damage prediction	●	●		●
Satellite-based oil spill detection	●	●		
Forecasting Ozone action days		●		
Air pollution-mortality link		●		
Predicting corporate bankruptcy	●	●		●
F-16 controls (ailerons)		●		
F-16 controls (elevators)			●	
Credit card fraud	●	●		

implementations are nondeterministic (second column): the same implementation, run repeatedly on the same dataset, e.g., "Credit card fraud", yields different clusterings. Second, different implementations of the same clustering algorithm are inconsistent (columns 3–5): different implementations run on the same dataset, e.g., "Pest damage prediction", or "Predicting corporate bankruptcy", yield different clusterings. When used in high-stakes domains, nondeterminism and inconsistency can have severe consequences.

1.2 Research Contributions

Prior research efforts have produced many clustering algorithms, and these algorithms have been implemented in numerous toolkits. But prior work has not questioned the clustering implementations' correctness or reliability. For example, developers use clustering optimistically assume that algorithms' implementations are correct, accurate, and generally reliable. However, ensuring clustering correctness, or even specifying clustering correctness, remain distant goals. Therefore, we propose differential clustering approaches to measure and validate the determinism and consistency across toolkits. This dissertation makes analytical contributions – statistical approaches for exposing the diverse root causes of nondeterminism or inconsistency: default parameter settings, noise insertion, distance metrics, termination criteria. This dissertation also makes practical contributions – an annotation and tracing-based approach to locate nondeterminism and inconsistency's root causes.

We introduce SmokeOut (Chapter 3), a tool that leverages the wide availability of clustering implementations and datasets with ground truth to test clustering implementations (while controlling for datasets and algorithms). Crucially, SmokeOut does not require an explicit specification associated with an implementation. SmokeOut uses a suite of differential clusterings coupled with a statistics-driven approach to help developers measure the determinism and accuracy (absolute, as well as relative to other toolkits) of a given implementation. In Section 3.4, we present the SmokeOut results. We now present a few highlights for our findings:

1. Deterministic algorithms have nondeterministic implementation across toolkits.

2. Nondeterministic algorithms have a wide range of outcomes: the variations across toolkits and variation across runs can be severe.

We also propose a statistically rigorous approach that couples differential clustering to with help developers (or toolkit testers) find statistically significant clustering accuracy differences. Our approach has four tests and we expose variations in a statistically rigorous way. Statistical tests show variations across runs shown in Section 4.1, variations across toolkits in Section 4.2 and variations across algorithms in Section 4.3.

We quantify implementation-induced nondeterminism and inconsistency of deterministic clustering algorithms, expose their root causes, and show how they can be alleviated, which in turn can improve both efficiency and effectiveness. Three algorithms are studied: Affinity Propagation (Section 5.2), DBSCAN (Section 5.3), and Hierarchical Agglomerative Clustering (Section 5.4).

This dissertation proposes an approach to automatically detect inconsistency (Section 6.2) of clustering algorithms via dynamic analysis, using an annotation framework (Section 6.3). This approach focuses on scikit-learn, R, and Elki toolkits.

1.3 Dissertation Outline

The remainder of this dissertation is organized as follows.

Chapter 2 presents background material, definitions, and the experimental setup. Clustering background is discussed in Section 2.1. Section 2.2 discusses clustering accuracy measures. Section 2.3 defines determinism and consistency formally. The widely-popular clustering toolkits we studied (MATLAB, MLpack, R, Scikit-learn, Shogun, TensorFlow, WEKA) are discussed in Section 2.4. The clustering algorithms used in our study are discussed in Section 2.5.

Section 3.1 presents SmokeOut, the first approach for differential testing of clustering implementations. To characterize clustering outcomes and present the results in an intuitive way, we introduce a concise, yet effective and statistically rigorous, 5-label system that captures distribution shapes (Section 3.3). Section 3.4 presents the SmokeOut results.

A statistically rigorous approach to testing clustering implementations is presented in Chapter 4: statistical tests on whether accuracy varies across runs (Section 4.1) and tests on variations across toolkits (Section 4.2). Section 4.3 shows the toolkit's impact when comparing algorithms and Section 4.4 tests on how different toolkits "disagree".

Chapter 5 exposes and explores the various causes of nondeterminism and inconsistency in deterministic clustering algorithms; we study 528 datasets, of which 400 are medical datasets (Section 5.1.1). Our quantitative analysis exposes several root causes of nondeterminism for three deterministic algorithm: Affinity Propagation

(Section 5.2), DBSCAN (Section 5.3), and Hierarchical Agglomerative Clustering (Section 5.4).

Chapter 6 proposes an approach to automatically detect inconsistency (Section 6.2) of clustering algorithms by using dynamic analysis with annotation framework (Section 6.3). We evaluate the effectiveness of our approach on Scikit-learn, R, and Elki in Section 6.4.

Chapter 7 reviews related work, comparing our approach with other cluster comparison studies as well as Machine Learning/Neural Networks reliability work.

Chapter 8 discusses several directions for future work.

CHAPTER 2

BACKGROUND

This chapter presents the definitions, metrics, datasets, toolkits, and clustering algorithms we used throughout the dissertation.

2.1 Clustering Definition

Given a set S of n points (d-dimensional vectors in the $\mathcal{R}^d$ space), a clustering is a partitioning of S into K non-overlapping subsets (clusters) $S_1, \dots, S_i, \dots, S_K$ such that intra-cluster distance between points (that is, within individual S_i's) is minimized, while inter-cluster distance (e.g., between centroids of S_i and S_j where $i \neq j$) is maximized [22].

2.2 Measuring Clustering Accuracy

Evaluating the performance of a clustering algorithm is not trivial (as compared to, for instance, measuring precision and recall for a supervised classification algorithm). There are a myriad metrics for comparing two clusterings (partitionings) C and C' of an underlying set D, e.g., *Mutual Information based scores* [51], *V-Measure* [46].

The *adjusted Rand index* (ARI), introduced by Hubert and Arabie [34] is an effective and intuitive measure of clustering outcomes: it allows two different partitioning schemes of an underlying set D to be compared. Multiple surveys and comparisons of clustering metrics have shown that ARI is the most widely used [49], most effective, as well as very sensitive [40]. Concretely, assuming two clusterings (partitionings) U and V of S, the ARI measures how similar U and V are. The ARI varies between -1 and $+1$, where $ARI = +1$ indicates perfect agreement, $ARI = 0$ corresponds to independent/random clustering, and $ARI = -1$ indicates "perfect disagreement", that is completely opposite assignment.

Per Vinh et al. [52], using the notation "N_{11} for the number of pairs that are in the same cluster in both U and V; N_{00} as the number of pairs that are in different clusters in both U and V; N_{01} as the number of pairs that are in the same cluster in U but in different clusters in V; and N_{10} as the number of pairs that are in different clusters in U but in the same cluster in V" [52] we have:

$$ARI(U,V) = \frac{2(N_{00}N_{11} - N_{01}N_{10})}{(N_{00}+N_{01})(N_{01}+N_{11}) + (N_{00}+N_{10})(N_{10}+N_{11})}$$

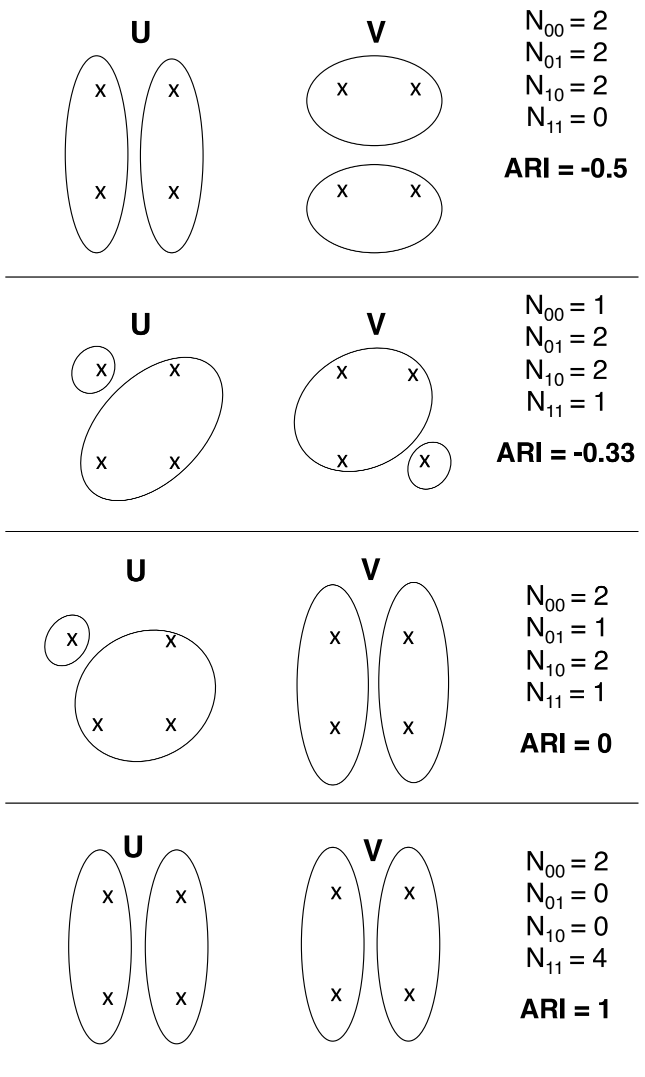

Figure 2.1 Different clusterings, U and V, of the same underlying 4-point dataset, and the resulting ARI.

In Figure 2.1 we illustrate how the same 4-point dataset can be clustered in four different ways, which leads to four different ARIs: -0.5, -0.33, 0, and 1, respectively. Notice how the ARI intuitively captures the similarity of various clusterings: in the top clusterings, U and V are strongly dissimilar – no two points are in the same cluster, hence the negative $\mathsf{ARI} = \mathsf{-0.5}$; in the second clustering, two points appear in the same cluster, hence the smaller dissimilarity $\mathsf{ARI} = \mathsf{-0.33}$; the third clustering, $\mathsf{ARI} = \mathsf{0}$, is usually called "independent"; finally, the fourth clustering is the "perfect agreement" case hence $\mathsf{ARI} = \mathsf{1}$.

2.3 Determinism and Consistency

In Table 1.1 we have presented two serious issues, nondeterminism and inconsistency, and defined them informally. We now proceed to defining them formally.

Determinism. A *deterministic* clustering of dataset D yields clustering solutions C'_R that are isomorphic to C_R. A clustering implementation, i.e., a toolkit implementing a deterministic algorithm, 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.

2.4 Toolkits

We now discuss the toolkits used in our studies and evaluation. We chose eight widely-used ML toolkits: MATLAB, MLpack, R, Scikit-learn, Shogun, TensorFlow, WEKA, Elki. The popularity of these toolkits is apparent in many ways: multi-million user bases, e.g., MATLAB and R; TensorFlow's 1,600+ GitHub contributors [4] or the abundance of S&P 500 companies that use TensorFlow [1]; Scikit-learn is used by popular services such as Spotify, Evernote, or Booking.com [13]; Pentaho

Corporation acquired an exclusive licence to use Weka for business intelligence; Elki has an emphasis on unsupervised methods in cluster analysis.

There are plenty of libraries or platforms supporting clustering analysis. Scikit-learn, Shogun and TensorFlow are Python libraries. R's clustering functions are developed in stat library, apcluster library and dbscan library. MATLAB has its own Statistics and Machine Learning Toolbox. MLpack is an intuitive, fast, and flexible C++ machine learning library. WEKA and Elki are Machine Learning and Data Mining tools that are developed on Java.

2.5 Clustering Algorithms

We presented the clustering algorithms used for testing in this section.

K**-means** [**?**] aims to cluster the observations (points in S) into K distinct clusters, where observations belong to the clusters with the nearest mean. The goal is to minimize the sum of all intra-cluster distances. The algorithm starts from K selected initial points as "centroids" (cluster centers); as we shall see, these centroids ultimately play a crucial role in the algorithm's effectiveness.

Figure 2.2 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 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.

***K*-means++** [5]was designed to improve K-means by choosing the starting points more carefully so they are farther apart. Theoretically, this improved version ensures that the algorithm is less likely to converge to local minima.

Gaussian/EM [20] aka Gaussian mixture clustering is a model-based approach: clustering is first done using a model (i.e., a parametric distribution such as a Gaussian). Each cluster is defined by an initial random model and the dataset is composed of the mixture of these models. Then, the model/data fitness is optimized – a common optimization is Expectation-Maximization.

Spectral clustering [36] computes eigenvalues of the similarity matrix between the data points to reduce the dimensionality of the original data. After the reduction, a clustering algorithm, e.g., K-means, is applied on the reduced-dimensionality data.

Hierarchical clustering [41] – 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).

Figure 2.3 shows the algorithm's pseudocode. Initially, each point d_i is its own cluster. Next, inter-cluster distances d(C_i,C_j) are computed and the closest clusters C_{mi},C_{mj} are merged. The algorithm continues until it reaches the desired number of clusters k.

DBSCAN [25] forms clusters by looking for "dense" regions, i.e., regions with at least minPoints separated by a maximum distance eps. DBSCAN's number is fixed: in the general scenario we explore here, it practically executes O(N^2)*steps*.

Figure 2.4 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.

Affinity Propagation (AP) [28] 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. AP proceeds in two phases: initialization followed by iteration. Figure 2.5 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.

Some toolkits do not support all seven algorithms; Table 2.1 shows the supported algorithm/toolkit combinations; in all, there were 33 algorithm-toolkit configurations.

Table 2.1 Toolkit/Algorithm Configurations

	MATLAB	MLpack	R	Scikit-learn	Shogun	TensorFlow	WEKA	Elki
kmeans++	✓	✓	✓	✓	✓	✓	✓	✓
kmeans	✓	✓	✓	✓	✓	✓		✓
spectral			✓	✓				
hierarchical	✓		✓	✓				✓
gaussian	✓			✓		✓	✓	✓
dbscan		✓	✓	✓				✓
apcluster			✓	✓				✓

2.6 Conclusion

In theory, implementations of deterministic clustering 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. In the rest of the dissertation we study these issues and propose solutions.

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


 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
         $m = \text{argmin} \sum_{j=1}^{K} \|x_i - C_j\|$
         $S_m = S_m \cup \{x_i\}$
     }
     // Update step: recompute centroids to be the mean of the updated clusters
     for (j=1; j <= $K$; j++) {
         $C_m = (\sum_{l=1}^{|C_m|} x_{lm})/|C_m|$
     }
       iter ++;
}
while (( clusters still changing || objective > minObjective)
         && iter < MAX_ITER);
```

Figure 2.2 K-means pseudocode.

```
procedure HierarchicalAgglomerativeClustering(D,k) :

// D = {$d_1$,$d_2$,...,$d_n$}

// start  with N singleton  clusters
$C$ = {{$\mathrm{d}_1$},{$\mathrm{d}_2$},...,{$\mathrm{d}_n$}};

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

return  list  of  clusters  $C$;
```

Figure 2.3 Hierarchical Agglomerative pseudocode.

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

$C = \emptyset$;

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 $\cup$ N';
      }
    }
  }
}

return list of clusters $C$;
```

Figure 2.4 DBSCAN pseudocode.

```
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 ($d(C, C^{new}) < \varepsilon$)
     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 $C$;
```

Figure 2.5 Affinity Propagation pseudocode.

CHAPTER 3

THE SMOKEOUT CLUSTERING TESTBED

To investigate the assumption that clustering implementations are deterministic and consistent (and consequently, toolkits implementing the same algorithm are interchangeable), in this chapter we introduce the SmokeOut approach and tool (testbed). SmokeOut uses a suite of differential clusterings coupled with a statistics-driven approach to help developers measure the determinism, accuracy, and relative accuracy of clustering toolkits (implementations).

3.1 SmokeOut Architecture

Crucially, SmokeOut does not require an explicit specification associated with an implementation, as it uses a suite of differential clusterings coupled with accuracy

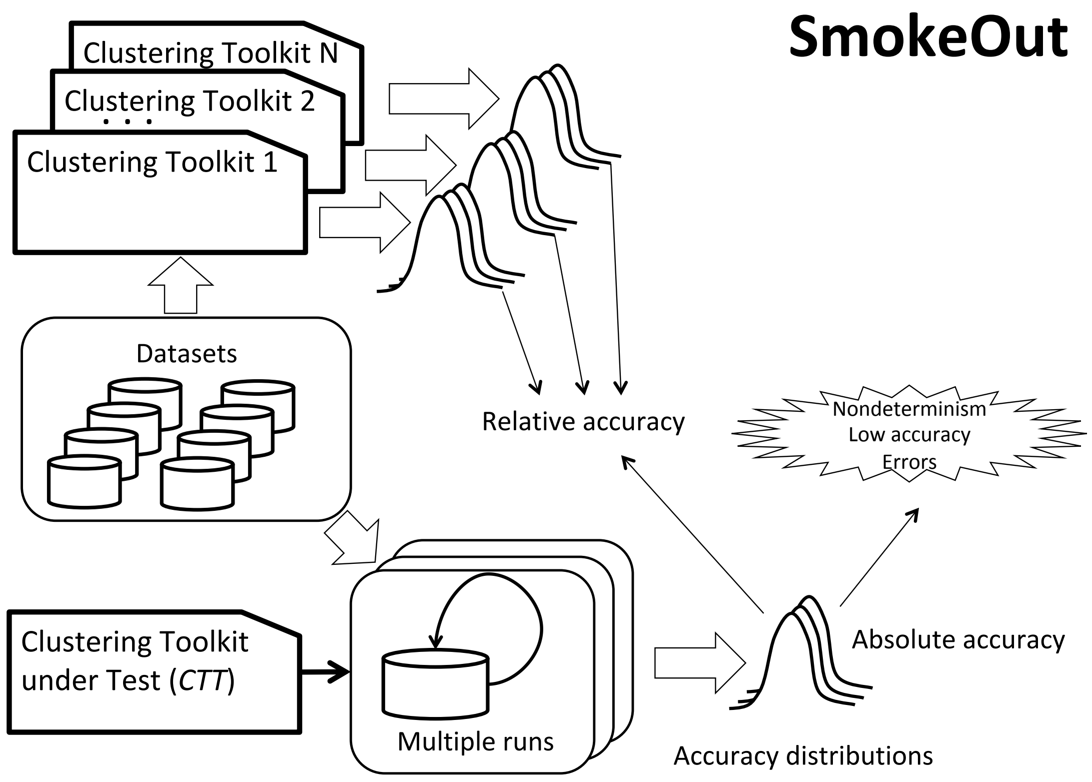

Figure 3.1 SmokeOut architecture.

distributions to automatically quantify issues such as nondeterminism and inconsistency. SmokeOut can also be used to detect low-accuracy anomalies; for this, it leverages the wide availability of datasets that come with “ground truth” (e.g., as available from classification repositories). More generally, SmokeOut can be used for a wide range of test scenarios for clustering implementations (while controlling for datasets and algorithms).

Figure 3.1 shows SmokeOut’s architecture. Let CTT be a new “Clustering Toolkit under Test” (bottom left of the figure), that is, an implementation of a specific clustering algorithm. CTT is tested as follows:

1. CTT is run multiple times on the same dataset to gauge (potential) nondeterminism. For this we run statistical analyses on the accuracy distributions.

2. CTT’s accuracy distributions are compared with other implementations of CTT’s algorithm (“Clustering Toolkit 1” ... “Clustering Toolkit N”); which allows us to measure CTT’s relative accuracy, or compare accuracy when ground truth is not available.

3.2 Datasets

We chose PMLB (Penn Machine Learning Benchmark), a benchmark suite that includes “real-world, simulated, and toy benchmark datasets” [44]. PMLB was designed to benchmark ML implementations and avoid imbalance across meta-features (which often plagues handpicked datasets). PMLB is a collection of 166 datasets, of which we used 162; we excluded connect-4, poker, mnist, and kddcup due to their excessive size – running these hundred of times would be prohibitive. The following table contains descriptive statistics for the 162 datasets.

Datasets have, on average, 809 instances (that is, points to be clustered) and the mean number of features (the number of attributes, or dimensions d) is 15. PMLB

Table 3.1 Descriptions for the PMLB Datasets

	Min	**Max**	**Geom. mean**
Instances	32	105,908	809.25
Features (attributes)	2	1,000	15.41
K (# of clusters)	2	26	3.18

comes with ground truth, which allows us to measure clustering accuracy. About half the datasets have two clusters ($K = 2$), while for the rest we have $3 \leq K \leq 26$.

Table 3.2 Categories for the PMLB Datasets

Category	**Percentage**
Medical/Health	24%
Biology, Biochemistry, Bioinformatics	15%
Physics, Math, Astronomy	11%
Social, Census	10%
Sports	7%
Financial	7%
Image recognition	6%
Synthetic datasets	6%
IT, AI	4%
Linguistics	3%
Miscellaneous	7%

We categorize the nature of each dataset and present the category breakdown in Table 3.2. We point out several things: the datasets are quite representative, as they cover a wide range of domains, from scientific to social to financial; medical data (discussed next) has the highest proportion, 24%; and the presence of synthetic datasets, 6%, to increase the variety of data density distributions.

To illustrate the need for clustering reliability, we note that 38 of the real-world datasets in PMLB are clustering tasks from the medical/health domain, e.g., contain patient data and outcomes. For example, four datasets are dedicated to breast cancer; three are focused on heart disease; other datasets involve predicting diabetes, hypothyroidism, appendicitis, etc.

3.3 Distribution Shapes

Since clustering implementations are used as "black boxes", SmokeOut users can also get an idea of what to expect from a certain toolkit or algorithm: *will clustering performance be consistently good? will it be consistently bad? will it be mostly good with an occasional "bad" run? will it be mostly bad with an occasional "good" run? will it be good for half the runs, and bad for the other half?*

There are many statistical parameters that characterize a distribution, but no single parameter to give us the answers to the previous questions. To this end, we introduce a simple, concise, five-label system that can succinctly characterize a distribution along the lines drawn above. The labels capture distribution shapes (illustrated in Figure 3.2 and described shortly) and are defined as follows:

R, which stands for outliers to the Right of the distribution; that is, clustering accuracy can sometimes be high; put prosaically, some runs are "good".

L, which stands for outliers to the Left of the distribution; that is, clustering accuracy can sometimes be low; put prosaically, some runs are "bad".

LR, when both good and bad outliers exist.

B, which stands for Bimodality – the distribution is bimodal, where a set of values if low and one is high.

U, aka Uniform values – no outliers.

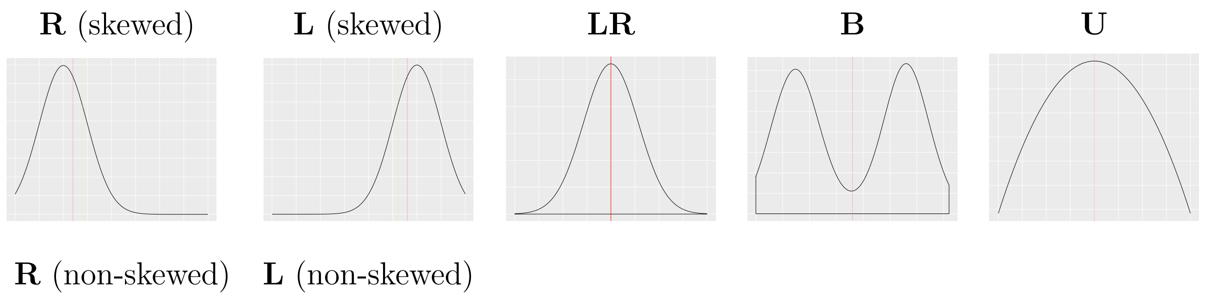

Figure 3.2 Distribution shapes and their corresponding labels; the red dotted vertical line indicates the median while the yellow dotted line is the mean.

Figure 3.2 shows these distribution shapes. For **L** and **R** we have two cases – when the distribution is skewed (top) and non-skewed (bottom) which will require us to be careful with outlier detection. The rest of the shapes, **LR**, **B**, and **U** are only applicable when the distribution is non-skewed (hence a single corresponding figure).

This five-label system, along with the minimum & maximum accuracy attained in our experiments serve as effective indicators of expected clustering performance over repeated runs: they show whether the algorithm is stable and accurate. We now proceed to define the statistical underpinnings of the label system: we first check for a bimodal distribution; if the distribution is not bimodal, we test for outliers.

3.3.1 Bimodality (B)

We assign the label **B** when the underlying distribution is bimodal. We use the *bimodality coefficient* [47]:

$$b = \frac{g^2 + 1}{k + \frac{3(n-1)^2}{(n-2)(n-3)}}$$

where n is the number of items in the sample, g is the skewness and k is the kurtosis. Intuitively, a higher b indicates a bimodal distribution, whereas a lower b indicates a unimodal one; we use a threshold value of 0.45, i.e., when $b > 0.45$ we declare the distribution bimodal.

3.3.2 Outliers (R, L, LR)

When the distribution is not bimodal, we need a statistical measure to detect outliers, even in the presence of skewness. We start with *Medcouple* (MC): introduced by Brys et al. [17], MC is a simple yet robust single parameter that can characterize distribution skewness – its shape and asymmetry. The MC captures the distribution's "tilt": negative MC's indicate left-skewed distributions while positive MC's indicate right-skewed distributions. For practical reasons we use $-0.1 \leq MC \leq 0.1$ as indicator of a symmetric distribution; we consider MC values lower than -0.1 (or greater that 0.1) to indicate left-skewness (or right-skewness, respectively).

We use the standard notations: Q_1 is the first quartile (25% percentile), Q_3 is the first quartile (75% percentile); IQR (interquartile range) is $IQR = Q_3 - Q_1$; range r is $r = Max - Min$. The table below indicates the adjusted ranges, i.e., outlier thresholds, depending on skewness:

Skewness	***Left threshold***	***Right threshold***
Left	Q_1	$Q_3 + 1.5e^{4MC}IQR$
Right	$Q_1 - 1.5e^{-4MC}IQR$	Q_3
Non-skewed	$Max - 0.8r$	$Min + 0.8r$

For the non-skewed case, the thresholds allow us to detect the lowest 20% "bad" outliers and the highest 20% "good" outliers. For skewed distributions, the outlier thresholds have to be moved to the right (for left-skewed distributions) and left, respectively (for right-skewed distributions) to account for the shift in the "bulk" of the distribution and avoid declaring bulk points as outliers. A vertical comparison between the skewed and non-skewed illustrations (Figure 3.2, the four graphs on the left) makes this point.

We detect outliers, and assign labels, as follows. If there are points that lie to the left of the adjusted range (that is, lower than the Left threshold), we assign label **L**. If there are points that lie to the right of the adjusted range (that is, higher than the Right threshold), we assign label **R**. If there are points that lie both to the left and the right of the adjusted range, we assign label **LR**. In other words, we treat tailed data as outliers.

Finally, if there are no outliers, we use the **U** label – that is, performance is expected to be stable/quasi-constant/uniform.

3.4 Results

3.4.1 SmokeOut Methodology

SmokeOut was run 30 times for each algorithm, so we can draw meaningful statistical conclusions; we used default settings for all toolkits. In all, across all algorithms and toolkits, there were 152,276 runs. We use the following format: for each of these 30 runs, we obtain 30 clustering outcomes. We compare these clusterings against Ground Truth, and measure the ARI. Next, we characterize the ARI distribution by indicating the *min* value, the *max* value, and the *shape* (that is, one of **B**, **R**, **L**, **LR**, **U**). Let us take the first row of Table 3.3 as an example, where K-means was run on dataset collins using SKlearn, R, MLpack, MATLAB, Shogun and TensorFlow. For SKlearn, across the 30 runs, we observed a minimum accuracy of 0.54, a maximum accuracy of 0.7, and the distribution shape is **LR** (both left and right outliers). That is, the expected accuracy is in the interval $[0.54, 0.7]$, with both left and right outliers possible. The next row, confidence, however, has a bimodal distribution with minimum 0.36 and maximum 0.71 hence running the toolkit repeatedly will yield accuracy values either in the neighborhood of 0.36 or in the neighborhood of 0.71, i.e., a $2\times$ variation from run to run! Note that the numbers discussed so far compare the clustering outcome against Ground Truth. The final set of columns, "T.A." (toolkit agreement), indicates the ARI when comparing toolkits against each other; in other words, we want to know if toolkits agree with each other (though they might disagree with Ground Truth). This is computed pairwise, toolkit vs. toolkit, hence we have $30 \times 30 = 900$ comparisons per toolkit pair. Based on these comparisons, we compute the ARI and indicate the minimum ARIs, as well as the toolkit combinations.

Finally, each table has three sets of rows: 25 "regular" rows on top where we show results for the top-25 *highest* accuracy for that algorithm. The *Median* and *Geometric Mean* are computed over the 25 datasets shown in the table. The last two rows, *Median (all)* and *Geometric Mean (all)*, are computed over all 162 datasets.

3.4.2 K-means with Varying Starting Points

The K-means algorithm requires "starting points", that is, initial cluster centers – with different starting points, the algorithm may converge to different minima. We explored the variation in outcome by randomly picking different starting points from the dataset to compare difference between toolkits.

Specifically, in each run we pick K (according the number of clusters in the dataset) points from the datasets and use them as initial centroids (cluster centers). We ran R in the default configuration (the 'R' grouped columns) as well as 100-iterations configuration ('R100iter' grouped columns) because by default R stops iterations early. Table 3.3 shows the results; we now proceed to discuss the results.

A bad choice of starting point can be *worse than random*. Note the fifth row, house-votes-84: all toolkits except R have a minimum value of *-0.02* (recall that $ARI = 0$ corresponds to random clustering); moreover, these toolkits' distributions are bimodal (marked with **B**) meaning the minimum is not an outlier (which would be marked as **L**).

MAX performance (best case). No toolkit outperforms consistently. For example, on dataset flags, *all toolkits except* R100iter have max accuracy of 0.11–0.15.

Table 3.3 K-Means: Variation Due To Starting Points

Dataset	**SKlearn**			**R**			**R100iter**			**MLpack**			**Matlab**			**Shogun**			**TensorFlow**			**T.A.**	
	Min	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Algo***
collins	.54	.70	LR	.56	.65	B	.54	.70	LR	.54	.70	LR	.54	.70	LR	.54	.70	LR	.54	.70	LR	.50	ALL
confidence	.36	.71	B	.36	.71	B	.36	.71	B	.36	.71	B	.36	.71	B	.36	.71	B	.36	.71	B	.30	ALL
corral	0	.38	B	0	.38	B	0	.38	B	-.01	.38	B	0	.38	B	0	.38	B	0	.38	B	-.08	ALL
ecoli	.47	.70	B	.47	.69	B	.47	.70	B	.47	.70	B	.47	.70	B	.47	.70	B	.47	.70	B	.51	ALL
house-votes-84	-.02	.54	B	.52	.54	U	-.02	.54	B	-.02	.54	B	-.02	.54	B	-.02	.54	B	-.02	.54	B	0	ALL
iris	.41	.73	B	.41	.78	B	.41	.73	B	.41	.73	B	.41	.73	B	.41	.73	B	.41	.73	B	.41	ALL
mfeat-karhunen	.48	.76	LR	.47	.76	R	.48	.76	LR	.48	.76	LR	.48	.76	LR	.48	.76	LR	.48	.76	LR	.42	ALL
mfeat-pixel	.50	.78	LR	.51	.78	LR	.50	.78	LR	.50	.78	LR	.50	.78	LR	.50	.78	LR	.50	.78	LR	.41	MT/R
monk3	.09	.39	B	.09	.39	B	.09	.39	B	.09	.39	B	.09	.39	B	.09	.39	B	.09	.39	B	-.01	ALL
mushroom	0	.37	B	0	.37	B	0	.37	B	0	.37	B	0	.37	L	0	.37	L	0	.37	L	-.01	ALL
new-thyroid	.16	.60	B	.21	.60	B	.16	.60	B	.16	.60	B	.16	.60	B	.16	.60	B	.16	.60	B	.14	ALL
optdigits	.57	.75	LR	.52	.75	LR	.57	.67	B	.57	.75	LR	.57	.75	LR	.57	.75	LR	.57	.75	LR	.55	MT/R/R1/S/SH/T
promoters	1	1	U	1	1	U	1	1	U	1	1	U	1	1	U	1	1	U	1	1	U	1	ALL
shuttle	.17	.45	B	.13	.45	LR	.24	.45	B	.24	.45	B	.24	.45	B	.24	.45	B	.24	.45	B	.19	MT/R/R1/SH/T
solar-flare_1	.08	.45	LR	.08	.45	LR	.08	.45	LR	.08	.45	LR	.08	.45	LR	.08	.45	LR	.08	.45	LR	.19	ALL
solar-flare_2	.12	.54	LR	.12	.54	LR	.12	.54	LR	.12	.57	LR	.12	.54	LR	.12	.54	LR	.12	.54	LR	.24	ALL
waveform-40	.25	.46	B	.25	.25	U	.25	.25	U	.25	.46	B	.25	.46	B	.25	.46	B	.25	.46	B	.37	ALL
soybean	.29	.49	LR	.29	.44	B	.29	.49	B	.29	.49	LR	.29	.49	LR	.29	.49	LR	.29	.49	LR	.47	ALL
satimage	.28	.52	B	.30	.53	B	.28	.52	B	.28	.52	B	.28	.52	B	.28	.52	B	.28	.52	B	.38	MP/MT/R/R1/SH/T
pendigits	.48	.62	B	.47	.62	B	.48	.62	B	.48	.62	B	.48	.62	B	.48	.62	B	.48	.62	B	.52	MT/R/R1/S/SH/T
mofn-3-7-10	-.06	.38	B	-.06	.38	B	-.06	.37	B	-.06	.38	B	-.06	.38	B	-.06	.38	B	-.06	.38	B	-.04	ALL
mfeat-pixel	.50	.78	LR	.51	.78	LR	.50	.78	LR	.50	.78	LR	.50	.78	LR	.50	.78	LR	.50	.78	LR	.41	MT/R
led7	.31	.49	LR	.31	.50	LR	.23	.49	LR	.23	.49	LR	.22	.49	LR	.31	.49	LR	.22	.49	LR	.23	MP/MT/R1/S/SH/T
haberman	-.01	-.01	U	-.01	.01	U	-.01	-.01	U	-.01	.17	B	-.01	-.01	U	-.01	-.01	U	-.01	-.01	U	-.01	ALL
flags	-.01	.15	B	0	.11	LR	-.01	.35	B	-.02	.15	B	-.01	.15	B	-.01	.15	B	-.01	.15	B	-.04	MP/MT
dna	.32	.45	B	.24	.47	LR	.33	.45	B	.32	.45	B	.32	.45	B	.32	.45	B	.32	.45	B	.54	MT/R
balance-scale	.04	.29	LR	.04	.29	LR	.04	.29	LR	.04	.33	LR	.04	.29	LR	.04	.29	LR	.04	.29	LR	-.01	ALL
Median	*.28*	*.52*		*.29*	*.53*		*.25*	*.52*		*.25*	*.52*		*.25*	*.52*		*.28*	*.52*		*.25*	*.52*		*.30*	
Geometric Mean	*.25*	*.52*		*.27*	*.51*		*.25*	*.52*		*.25*	*.53*		*.25*	*.52*		*.25*	*.52*		*.25*	*.52*		*.26*	
Median (all)	*.01*	*.07*		*.01*	*.07*		*.01*	*.07*		*.01*	*.07*		*.01*	*.07*		*.01*	*.07*		*.01*	*.07*		*.30*	
Geometric Mean (all)	*.09*	*.16*		*.09*	*.16*		*.09*	*.16*		*.09*	*.17*		*.09*	*.16*		*.09*	*.16*		*.09*	*.16*		*.26*	

However, R100iter achieves three times better accuracy: .35! For haberman, with the exception of MLpack, all toolkits' max hovers around 0; hence, except MLpack, all toolkits' top performance is close to random.

MIN performance (worst case). house-votes-84 shows the danger of local minima: all toolkits, except R, have mins of -0.02 (worse than random). Occasionally, these toolkits will achieve high accuracy. However, R users are much better "protected": their min is essentially the same as their max (.52 min, .54 max).

Table 3.4 *K*-means++

Dataset	SKlearn			R			R100iter			MLpack			Matlab			Weka			Shogun			TensorFlow			T.A.	
	Min	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Algo***
collins	.56	.65	B	.55	.65	B	.57	.65	B	.54	.70	B	.65	.70	U	.27	.38	LR	.50	.70	LR	.49	.64	LR	.20	MP/W
confidence	.58	.58	U	.58	.65	B	.58	.61	U	.36	.71	B	.57	.69	B	.39	.71	LR	.35	.71	B	.36	.71	LR	.05	MP/T
corral	.13	.31	B	-.01	.31	B	.13	.31	B	-.01	.31	B	.13	.18	B	-.01	.37	B	-.01	.34	B	-.01	.38	B	-.09	MP/SH/W
ecoli	.46	.53	B	.46	.53	B	.47	.53	B	.42	.70	B	.68	.70	U	.44	.72	B	.42	.70	B	.44	.72	B	.43	MP/W
house-votes-84	.54	.54	U	.54	.54	U	.54	.54	U	.01	.54	B	.54	.54	U	-.02	.54	B	-.02	.54	B	-.02	.54	B	-.01	MP/SH/T/W
iris	.73	.73	U	.73	.73	U	.73	.73	U	.41	.73	B	.73	.73	U	.42	.71	B	.41	.73	B	.71	.73	U	.42	MP/SH/W
mfeat-karhunen	.56	.76	LR	.55	.70	LR	.56	.75	LR	.50	.70	LR	.55	.66	B	.45	.64	LR	.50	.75	LR	.51	.74	B	.35	MP/W
mfeat-pixel	.55	.69	B	.56	.69	B	.56	.75	LR	.48	.67	LR	.57	.61	U	.49	.69	LR	.49	.75	LR	.52	.72	B	.41	MP/W
monk3	.09	.09	U	.09	.09	U	.09	.09	U	-.01	.39	B	.07	.09	U	-.01	.23	B	-.01	.39	B	.01	.09	B	-.01	SH/T
mushroom	.11	.11	U	.11	.11	U	.11	.11	U	0	.11	B	.05	.11	B	0	.23	B	0	.11	B	0	.37	B	-.07	T/W
new-thyroid	.24	.62	B	.57	.59	U	.57	.59	U	.16	.60	B	.16	.59	B	.43	.62	B	.16	.59	B	.16	.62	B	.13	MP/MT/SH/T/W
optdigits	.66	.67	U	.67	.67	U	.67	.67	U	.52	.75	B	.59	.67	B	.50	.76	LR	.57	.75	LR	.57	.75	LR	.44	SH/W
promoters	1	1	U	1	1	U	1	1	U	1	1	U	1	1	U	-.01	.40	B	1	1	U	1	1	U	-.01	ALL
shuttle	0	0	U	.25	.27	U	.16	.32	LR	.14	.37	LR	0	.26	B	.18	.27	B	.24	.41	B	0	.45	B	-.01	MP/T
solar-flare_1	.26	.44	B	.25	.28	U	.25	.45	B	.07	.33	B	.22	.25	U	.03	.17	LR	.07	.45	LR	.06	.45	LR	-.01	SH/W
solar-flare_2	.33	.56	LR	.25	.57	LR	.12	.55	LR	.10	.48	LR	.15	.28	B	.07	.16	LR	.09	.55	LR	.13	.57	B	.04	SH/W
vote	.57	.58	U	.58	.58	U	.58	.58	U	0	.58	B	.57	.58	U	.57	.58	U	.57	.58	U	.57	.58	U	-.01	MP/S/SH/T/W
spambase	.03	.03	U	.03	.03	U	.03	.03	U	.03	.03	U	.03	.03	U	-.03	.35	B	.03	.03	U	0	.03	U	-.06	ALL
dermatology	.02	.02	U	.02	.02	U	.02	.02	U	.01	.06	U	.01	.06	U	.31	.91	B	.01	.08	B	.01	.06	B	-.02	MP/W
crx	0	0	U	0	0	U	0	0	U	0	0	U	0	0	U	0	.50	B	0	0	U	0	0	U	-.01	MP/MT/S/SH/T/W
credit-a	0	0	U	0	0	U	0	0	U	0	0	U	0	0	U	-.01	.50	B	0	0	U	0	0	U	-.01	MP/S/SH/T/W
buggyCrx	0	0	U	0	0	U	0	0	U	0	0	U	0	0	U	0	.50	B	0	0	U	0	0	U	-.01	MP/MT/S/SH/T/W
australian	0	0	U	0	0	U	0	0	U	0	0	U	0	0	U	-.01	.50	B	0	0	U	0	0	U	-.01	MP/MT/SH/T/W
appendicitis	.31	.33	U	.31	.31	U	.31	.31	U	-.06	.37	B	.29	.29	U	-.06	.37	B	.29	.37	B	.29	.37	B	.07	MP/SH/T/W
Median	*.28*	*.48*		*.28*	*.42*		*.28*	*.49*		*.05*	*.44*		*.19*	*.28*		*.05*	*.50*		*.12*	*.49*		*.10*	*.50*		*-.01*	
Geometric Mean	*.29*	*.35*		*.31*	*.36*		*.31*	*.37*		*.17*	*.39*		*.28*	*.34*		*.16*	*.48*		*.21*	*.41*		*.21*	*.41*		*.08*	
Median (all)	*.02*	*.05*		*.02*	*.05*		*.03*	*.05*		*.01*	*.06*		*.02*	*.05*		*0*	*.15*		*.01*	*.07*		*.01*	*.07*		*-.01*	
Geometric Mean (all)	*.12*	*.14*		*.12*	*.14*		*.12*	*.14*		*.08*	*.15*		*.11*	*.14*		*.08*	*.20*		*.09*	*.16*		*.09*	*.16*		*.08*	

Instability, as revealed by distribution shapes. Recall that **U** indicates "predictable" performance. However, the table shows the abundance of bimodality and outlier-prone outcomes, i.e., **B**, **L**, **R**, **LR**. The last row shows the median values for min- and max- accuracy, respectively. Notice how accuracy can vary from .25–.29 (min) to .52–.53 (max), indicating a large degree of instability.

3.4.3 *K*-means++

Table 3.4 shows the clustering outcomes when running *K*-means with starting points generated according the *K*-means++ initialization algorithm. However, for *K*-means++ we do not control how the starting points are chosen, as *K*-means++

is supposed to improve upon K-means with a better initialization. We now discuss the findings.

No real improvement compared to random starting points. Despite the fact that K-means++ was devised to improve upon K-means, in our experiments K-means++ does not achieve higher accuracy compared with the random starting points (Section 3.4.2). Indeed, in the last rows, showing the median values for min- and max- accuracy, respectively, we observe that we are around the same values: .22–.31 to .29–.54. However, there is an improvement in terms of stability – comparing the shape labels in Table 3.3 and those in Table 3.4 we see more stability (more **U**'s).

Weka differs from the other toolkits. When using K-means++, we were able to add Weka to our study (Weka does not permit specifying starting points hence its absence from Section 3.4.2). Weka has interesting behavior, markedly different from the other toolkits. For example, if we look at credit-a through australian datasets (lower half of the table) we can see that no algorithm can break 0 (they achieve 0 min/max with a uniform distribution) whereas Weka has a bimodal distribution with accuracies of up to 0.5. Unfortunately, for the "easy" promoters set where all algorithms achieve a 1 score, Weka can only manage between 0.1 and 0.41.

Similarly, the agreement columns show that Weka has, in some cases, minimum agreement with other toolkits (e.g., solar-flare_1, but it is never present when agreeing with the maximum values. Even worse, on a large number of cases (not shown due to space limits), all toolkits report a 100% agreement with each other except with Weka!

We reached out to WEKA developers who suggested that we change WEKA's default configuration (turn normalization off) to improve its performance on this particular dataset [8]. While turning off normalization improved the performance on this dataset. We believe it is important for uniformity to run all toolkits with default parameters, as per-dataset tweaking might affect behavior negatively for other datasets.

MAX performance (best case). Similarly to K-means with random starting points, for K-means++ no toolkit outperforms consistently. For example, on monk3, Shogun and MLPack have a maximum accuracy of 0.39 where other algorithms do not get higher than 0.09 (with the exception of Weka which have a maximum score of 0.23), that is four times lower!

MIN performance (worst case). The minimum shows that even if considering using a specific algorithm for drawing our starting points, the difference min/max can be important. MLPack seems to be really sensitive as its min/max can range greatly. A clear examples is solare_flare_2 with a minimum of 0.1 where the maximum was 0.49; similarly for vote where minimum is 0.01 and maximum is 0.59.

3.4.4 Hierarchical (Agglomerative)

Table 3.5 shows the results obtained with the hierarchical (agglomerative) clustering.

Deterministic runs. Unlike the previous algorithms, hierarchical is deterministic, hence we expect no variation between runs. Indeed we find that for a given toolkit, distribution is uniform (all **U**'s).

Difference across toolkits. What is concerning however, is the difference between toolkits, e.g., on sets house-votes-84 (max is .59 for SKLearn, .67 for R, .33 for Matlab) or balance-scale (max's were .16, .17, and .12 respectively). For a deterministic algorithm there should be no such variation.

Toolkit (dis)agreement. Excepting some specific cases, all toolkits agree on their outcome as we have a large number of 1 as minimum values. A few datasets however show some disagreement, e.g., car and solar-flare_1. However, for a deterministic algorithm, there should be no such disagreement.

Table 3.5 Hierarchical

Dataset	SKlearn			R			Matlab			T.A.	
	Min	*Max*	*Shape*	*Min*	*Max*	*Shape*	*Min*	*Max*	*Shape*	*Min*	*Algo*
Hill_Valley_with_noise	0	0	U	0	0	U	0	0	U	1	ALL
analcatdata_germangss	.01	.01	U	.01	.01	U	.01	.01	U	1	ALL
balance-scale	.16	.16	U	.17	.17	U	.12	.12	U	.29	MT/S
vote	.49	.49	U	.49	.49	U	.49	.49	U	1	ALL
breast-cancer-wisconsin	.28	.28	U	.28	.28	U	.28	.28	U	1	ALL
analcatdata_aids	-.02	-.02	U	-.02	-.02	U	-.02	-.02	U	1	ALL
cmc	.01	.01	U	.03	.03	U	.01	.01	U	.46	R/S
analcatdata_happiness	.10	.10	U	.10	.10	U	.10	.10	U	1	ALL
backache	-.01	-.01	U	-.01	-.01	U	-.01	-.01	U	1	ALL
buggyCrx	0	0	U	0	0	U	0	0	U	1	ALL
analcatdata_japansolvent	0	0	U	0	0	U	0	0	U	1	ALL
mfeat-karhunen	.57	.57	U	.57	.57	U	.57	.57	U	1	ALL
ionosphere	.18	.18	U	.18	.18	U	.18	.18	U	1	ALL
car	0	0	U	0	0	U	-.01	-.01	U	.17	MT/R
solar-flare_1	.25	.25	U	.25	.25	U	.24	.24	U	.61	R/S
pima	.10	.10	U	.10	.10	U	.10	.10	U	1	ALL
house-votes-84	.59	.59	U	.67	.67	U	.33	.33	U	.36	MT/S
allhypo	.02	.02	U	.02	.02	U	.02	.02	U	1	ALL
tic-tac-toe	0	0	U	0	0	U	-.02	-.02	U	-.08	MT/R
pendigits	.55	.55	U	.55	.55	U	.55	.55	U	.99	ALL
waveform-21	.31	.31	U	.31	.31	U	.31	.31	U	1	ALL
analcatdata_asbestos	.11	.11	U	.11	.11	U	.11	.11	U	1	ALL
flags	.02	.02	U	.02	.02	U	.03	.03	U	.97	ALL
soybean	.40	.40	U	.38	.38	U	.38	.38	U	.82	MT/S
analcatdata_authorship	.76	.76	U	.77	.77	U	.77	.77	U	.94	ALL
Median	*.10*	*.10*		*.10*	*.10*		*.10*	*.10*		*1*	
Geometric Mean	*.20*	*.20*		*.20*	*.20*		*.19*	*.19*		*.79*	
Median (all)	*.04*	*.04*		*.03*	*.03*		*.02*	*.02*		*1*	
Geometric Mean (all)	*.13*	*.13*		*.13*	*.13*		*.12*	*.12*		*.72*	

Chapter 6 shows how we can automatically find the root causes of these determinism violations.

3.4.5 EM/Gaussian

Table 3.6 shows the results on Gaussian mixture.

MAX performance (best case). This algorithm stands out in that multiple toolkits achieve max performance of 0.9 or higher (Matlab, for instance, does so on four datasets, while SKlearn and Weka do so on three!).

MIN performance (worst case). house-vote-84 poses difficulties for all toolkits (minimum is around -0.03 to -0.01) except Weka, which achieves a minimum of 0.56!

The tolerance parameter from SKlearn has limited impact on results. Only for waveform-40, SKlearn with default tolerance attains a 0.25 max, whereas SKlearn0t attains double the accuracy (0.53).

Overall, best performance. EM/Gaussian is the only algorithm where multiple toolkits (Matlab and TensorFlow) exceed a 0.2 geometric mean across the 162 datasets.

3.4.6 Spectral

Table 3.7 shows the performance for SKlearn (Gaussian), SKlearnFast (k-nearest) and R (Gaussian).

SKlearnFast is a solid all-around choice. SKlearnFast outperforms other implementations, e.g., analcatdata_authorship reports a performance of .96 where it is around .63 to .72 for R and 0 for SKlearn! Similar for mfeat-pixel. More than having a high global performance, it is also quite stable (a lot of **U**'s) despite the fact that it is supposed to sacrifice accuracy in the name of efficiency (compared to Gaussian).
MAX performance (best case). The max values range from 0 to .92 depending on the dataset and the toolkit. However, SKlearn shows the worst performance as it

Table 3.6 EM/Gaussian

Dataset	SKlearn			SKlearn0T			Matlab			Weka			TensorFlow			T.A.	
	Min	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Algo***
prnn_crabs	0	0	U	.02	.02	U	-.01	.97	B	-.01	-.01	U	-.01	1	B	-.02	MT/T
analcatdata_creditscore	-.04	.26	B	-.05	.26	B	-.03	.95	B	0	0	U	-.03	.25	B	-.08	S/S0/T
twonorm	.90	.90	U	.90	.90	U	0	.91	B	.91	.91	U	-.01	.90	B	-.05	MT/T
analcatdata_authorship	.59	.90	LR	.50	.90	LR	.04	.79	LR	.95	.95	U	-.01	.41	B	-.09	MT/T
wdbc	.81	.81	U	.81	.81	U	0	.75	B	.67	.67	U	.03	.76	B	-.01	MT/T
breast-cancer-wisconsin	.81	.81	U	.81	.81	U	0	.72	B	.67	.67	U	.03	.76	B	-.01	MT/T
ionosphere	.39	.40	U	.39	.40	U	-.02	.43	B	.25	.25	U	0	.77	B	-.04	MT/T
wine-recognition	.45	.60	B	.44	.61	B	.32	.94	B	.91	.91	U	.02	.49	B	-.06	MT/T
breast	-.01	.70	B	-.01	.70	B	-.01	.58	B	.76	.76	U	.48	.67	B	-.01	MT/S/S0/W
dermatology	.08	.35	B	.02	.36	B	.41	.84	B	.52	.82	B	.11	.65	LR	-.06	MT/T
new-thyroid	.86	.86	U	.86	.86	U	.41	.90	B	.89	.89	U	.40	.86	LR	.24	MT/T
vote	.47	.54	B	.47	.54	B	0	.62	B	.47	.57	B	-.03	.45	B	-.03	MT/T
iris	.90	.90	U	.90	.90	U	.56	.56	U	.75	.75	U	.51	.90	B	.50	T/W
house-votes-84	-.02	.49	B	-.02	.49	B	-.03	.56	B	.55	.55	U	-.02	.57	B	-.05	MT/T
biomed	.18	.54	B	.18	.57	B	.01	.55	B	.54	.54	U	0	.57	B	-.02	MT/T
dna	.26	.50	LR	.33	.49	B	-.02	.10	LR	.30	.72	B	-.03	.01	U	-.03	T/W
waveform-40	.25	.25	U	.53	.53	U	.25	.56	B	.25	.25	U	0	.53	B	-.03	MT/T
ecoli	.27	.61	B	.25	.61	B	0	0	U	.34	.73	B	.53	.75	B	0	ALL
confidence	.32	.60	B	.32	.67	B	.30	.66	LR	.57	.75	B	.35	.62	B	.27	S0/T
promoters	1	1	U	1	1	U	-.01	.25	B	.45	.62	B	-.01	.03	U	-.05	MT/T
optdigits	.41	.57	B	.36	.61	LR	.45	.66	LR	.22	.61	B	.29	.53	LR	.12	T/W
waveform-21	.25	.25	U	.57	.57	U	.15	.58	B	.25	.25	U	.15	.57	B	.06	MT/T
splice	.02	.35	B	.02	.36	B	-.02	.26	B	.23	.34	LR	-.03	.49	B	-.07	MT/T
mushroom	0	.12	B	0	.38	B	-.01	.45	B	.07	.07	U	-.01	.49	B	-.10	MT/T
shuttle	.04	.23	B	.03	.20	B	.03	.50	B	.19	.32	B	.08	.27	B	.01	S/S0
Median	*.29*	*.55*		*.37*	*.59*		*0*	*.58*		*.46*	*.62*		*0*	*.57*		*-.03*	
Geometric Mean	*.35*	*.54*		*.37*	*.58*		*.09*	*.57*		*.44*	*.53*		*.09*	*.53*		*.01*	
Median (all)	*.01*	*.10*		*.01*	*.09*		*0*	*.16*		*.03*	*.10*		*-.01*	*.16*		*-.02*	
Geometric Mean (all)	*.10*	*.18*		*.11*	*.18*		*.04*	*.21*		*.13*	*.18*		*.03*	*.22*		*.01*	

is the only one to perform consistently worse than random (9 times, its max is around 0; across all datasets it has a min/max of 0, too), whereas the max for SKlearnFast and R are much higher.

SKlearnFast agrees more with R than with SKlearn. Last columns show that R generally agrees with SKlearnFast regarding maximums (18 times), where it generally agrees with SKlearn for the minimums (20 times).

3.4.7 DBSCAN

Recall that DBSCAN is deterministic; Table 5.7 shows the results.

Low performance (with default parameters). We noticed that DBSCAN can suffer from low accuracy with default parameters. For example, on datasets new-thyroid and analcatdata_lawsuit, accuracy can be as low as -0.2 and -0.1, respectively: *lower than random and lower than K-means++.* To gauge the impact of (small) variations in defaults, we also ran experiments where we varied its *minPoints* and ε parameters.[1] This leads to wide-spread bimodality and outliers – note the **B**'s and **LR**'s. For example, the accuracy varies widely for the same toolkit across different runs: this range can be as large as 0.95 (min 0, max 0.95) for analcatdata_creditscore, 0.89 (min -0.2, max 0.69) for new-thyroid or 0.84 for breast-w (min -0.07, max 0.77). This finding is especially worrisome for a deterministic algorithm.

3.4.8 Affinity Propagation

Table 3.9 shows the results. Recall that this algorithm (AP) is deterministic. AP uses a *dampingFactor* parameter.[2]

Variation across toolkits. Given that this algorithm is deterministic, we should not see variation across toolkits when toolkits are run with the same parameter (damping factor). However, max performance differed substantially, e.g., on breast-w max was .47 for SKLearn and .17 for R; for cleveland-nominal max was .31 for SKLearn and .03 for R.

[1]These control the minimum cluster size (our range was $1 \leq minPoints \leq 10$) and maximum neighborhood size (our range was $0 < \varepsilon < 10$.
[2]The factor controls oscillations; in our experiments $0.5 < dampingFactor < 1$.

Variation across runs. Our experiments show that the damping factor induces substantial differences between min and max performances across runs. This was the case for both SKLearn and R, e.g., for SKLearn we had collins (min .19, max .63) or breast-w (min .03, max .47) and for balance-scale on R we had (min 0.03, max .15).

3.5 Conclusion

We introduced SmokeOut, an approach for testing clustering implementations that leverages the current abundance of datasets and clustering toolkits. We applied SmokeOut to quantify clustering outcomes across different algorithms, toolkits, and multiple runs. Our findings exposed outliers and characterized distribution shapes. In Chapter 4 we show how we use statistical analysis of clustering outcomes across multiple runs, toolkits and algorithms to ensure statistical rigor.

Table 3.7 Spectral Clustering

Dataset	**SKlearn**			**SKlearnFast**			**R**			**T.A.**			
	Min	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Algo***	***Max***	***Algo***
breast-w	.10	.10	U	.80	.80	U	.05	.81	LR	-.03	R/S	.93	R/SF
mfeat-pixel	0	0	U	.92	.92	U	.55	.92	LR	-.01	R/S	.96	R/SF
dermatology	.01	.01	U	.17	.17	U	.47	.86	LR	-.04	R/S	.18	R/SF
breast-cancer-wisconsin	-.01	0	U	.41	.41	U	0	.53	B	-.01	R/S	.18	R/SF
wdbc	-.01	.32	B	.41	.41	U	.02	.53	B	-.01	R/S	.43	R/S
analcatdata_lawsuit	-.07	0	B	.03	.03	U	.31	.69	B	-.08	R/S	.05	R/SF
analcatdata_creditscore	-.05	.26	B	.84	.84	U	-.03	-.01	U	-.07	R/S	.19	S/SF
confidence	.01	.01	U	.70	.70	U	.32	.68	B	-.14	R/S	.86	R/SF
appendicitis	.35	.35	U	.46	.46	U	-.04	.45	B	-.09	R/SF	.76	R/SF
corral	.48	.48	U	.13	.13	U	-.01	.38	B	-.02	R/S	.55	R/SF
collins	-.01	0	U	.61	.63	U	.40	.66	LR	-.02	R/S	.60	R/SF
new-thyroid	-.12	-.08	U	.27	.27	U	.23	.50	B	-.12	R/S	.23	R/SF
mfeat-fourier	.54	.56	U	.56	.56	U	.41	.62	LR	.47	R/S	.66	R/S
mfeat-factors	-.01	0	U	.67	.67	U	.47	.66	B	-.01	R/S	.83	R/SF
mfeat-zernike	0	0	U	.56	.57	U	.47	.66	B	-.04	R/S	.80	R/SF
mfeat-morphological	0	.01	U	.23	.30	B	.17	.45	B	-.03	R/S	.37	R/SF
solar-flare_2	.08	.10	U	-.02	.04	B	.11	.41	B	-.10	R/SF	.61	R/SF
analcatdata_bankruptcy	-.02	.18	B	.45	.45	U	.04	.30	B	-.12	R/S	.43	R/S
iris	.74	.74	U	.75	.75	U	.55	.70	B	.58	R/S	.94	S/SF
ecoli	.60	.62	U	.50	.50	U	.58	.73	LR	.47	R/SF	.70	R/SF
balance-scale	-.01	.31	LR	.13	.13	U	0	.21	B	0	R/S	.80	S/SF
soybean	.01	.03	U	.19	.29	B	.25	.46	B	.01	R/S	.58	R/SF
threeOf9	-.01	.29	B	-.01	.12	B	-.01	.09	B	-.01	ALL	.51	S/SF
analcatdata_authorship	-.01	.01	U	.96	.96	U	.63	.72	B	-.01	S/SF	.75	R/SF
lupus	-.02	0	U	.19	.21	U	-.02	.25	B	-.04	R/S	1	R/SF
Median	*0*	*.03*		*.45*	*.45*		*.23*	*.53*		*-.02*		*.61*	
Geometric Mean	*.08*	*.15*		*.41*	*.43*		*.22*	*.51*		*.01*		*.57*	
Median (all)	*-.01*	*0*		*.03*	*.03*		*0*	*.04*		*-.01*		*.57*	
Geometric Mean (all)	*.03*	*.05*		*.12*	*.12*		*.07*	*.15*		*.02*		*.52*	

Table 3.8 DBSCAN

Dataset	**SKlearn**			**R**			**MLPack**			**T.A.**	
	Min	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Algo***
analcatdata_creditscore	0	.95	B	0	.95	B	0	.95	B	-.06	ALL
breast-w	-.07	.77	B	-.07	.77	B	-.07	.77	B	-.34	ALL
new-thyroid	-.20	.69	B	-.20	.69	B	-.21	.69	B	-.26	ALL
ionosphere	0	.65	B	0	.65	B	0	.66	B	-.12	ALL
collins	0	.63	B	0	.63	B	0	.63	B	-.08	ALL
iris	0	.56	B	0	.56	B	0	.56	B	0	ALL
vote	-.01	.47	B	-.01	.47	B	-.02	.45	B	-.12	ALL
spect	-.08	.32	B	-.08	.32	B	-.10	.32	B	-.17	ALL
analcatdata_lawsuit	-.10	.29	B	-.10	.29	B	-.10	.29	B	-.24	ALL
led7	0	.32	B	0	.32	B	0	.32	B	0	ALL
house-votes-84	-.03	.30	B	-.03	.30	B	-.02	.31	B	-.16	ALL
soybean	-.02	.27	B	-.02	.27	B	-.02	.30	B	0	ALL
mfeat-fourier	0	.29	B	0	.29	B	0	.29	B	0	ALL
titanic	0	.27	B	0	.27	B	0	.27	B	0	ALL
spectf	-.02	.26	B	-.02	.26	B	-.02	.26	B	0	ALL
prnn_fglass	0	.26	B	0	.26	B	0	.26	B	0	ALL
glass	0	.26	B	0	.26	B	0	.26	B	0	ALL
dermatology	0	.21	B	0	.21	B	0	.21	B	-.15	ALL
dna	-.06	.18	B	-.06	.18	B	-.06	.18	B	-.12	ALL
lymphography	0	.21	B	0	.21	B	-.04	.17	B	-.17	ALL
page-blocks	-.01	.19	LR	-.01	.19	LR	-.01	.20	LR	-.05	ALL
haberman	-.08	.16	LR	-.08	.16	LR	-.06	.16	B	-.22	ALL
agaricus-lepiota	-.01	.19	B	-.01	.19	B	-.01	.19	B	-.02	ALL
clean2	-.09	.15	LR	-.09	.15	LR	-.09	.15	LR	-.01	ALL
tic-tac-toe	0	.17	B	0	.17	B	0	.17	B	0	ALL
Median	*-.01*	*.27*		*-.01*	*.27*		*-.01*	*.29*		*-.06*	
Geometric Mean	*-.03*	*.35*		*-.03*	*.35*		*-.04*	*.35*		*-.10*	
Median (all)	*-.01*	*.01*		*-.01*	*.01*		*-.01*	*.01*		*-.01*	

Table 3.9 Affinity Propagation

Dataset	**SKlearn**			**R**			**T.A.**	
	Min	***Max***	***Shape***	***Min***	***Max***	***Shape***	***Min***	***Algo***
collins	.19	.63	B	.21	.62	B	.11	ALL
breast-w	.03	.47	B	.07	.17	B	.10	ALL
iris	.42	.67	B	.44	.64	B	.47	ALL
cleveland-nominal	-.02	.31	B	.02	.03	U	.11	ALL
tokyo1	-.01	.31	L	.10	.30	B	0	ALL
promoters	0	.28	B	.23	.28	U	0	ALL
mfeat-morphological	0	.28	B	0	.27	B	-.01	ALL
ecoli	.21	.37	B	.21	.24	U	.15	ALL
titanic	-.01	.24	B	-.01	.09	B	0	ALL
soybean	.21	.34	LR	.21	.25	U	.56	ALL
wine-recognition	.17	.30	B	.17	.21	U	.45	ALL
analcatdata_cyyoung9302	0	.19	B	.18	.19	U	0	ALL
analcatdata_creditscore	-.03	.16	LR	-.03	.16	LR	.01	ALL
dermatology	0	.15	B	.14	.15	U	.05	ALL
confidence	.24	.31	B	.24	.31	B	.50	ALL
new-thyroid	.04	.17	B	.12	.17	B	0	ALL
segmentation	0	.14	B	.12	.14	U	0	ALL
mfeat-fourier	0	.13	B	.12	.13	U	0	ALL
balance-scale	0	.06	LR	.03	.15	B	0	ALL
analcatdata_bankruptcy	.04	.15	B	.04	.15	B	.07	ALL
solar-flare_1	.08	.17	B	.09	.15	B	.13	ALL
prnn_synth	.07	.17	B	.07	.11	U	.38	ALL
solar-flare_2	.01	.13	B	.02	.12	B	0	ALL
prnn_fglass	.13	.20	B	.13	.17	U	.10	ALL
glass	.13	.20	B	.13	.17	U	.10	ALL
Median	*.02*	*.22*		*.12*	*.17*		*.07*	
Geometric Mean	*.07*	*.26*		*.12*	*.21*		*.12*	
Median (all)	*0*	*.04*		*.01*	*.03*		*.05*	

CHAPTER 4

STATISTICALLY RIGOROUS TESTING OF CLUSTERING IMPLEMENTATIONS

In Chapter 3, we exposed outliers and characterized distributions shapes. Descriptive statistics (min/max) were used to compare runs (as well as toolkits and algorithms), which is concise but lacks statistical rigor. In this chapter, we introduce and use a statically rigorous approach for comparing runs, toolkits, and algorithms.

4.1 Variation Across Runs

In this section, we test a null hypothesis on how different runs of the same algorithm in the same implementation lead to different clusterings.

This testing procedure is shown in Figure 4.1: a single toolkit is run on a single dataset multiple times (30 in our case), and a statistical analysis is performed on the resulting accuracy distribution.

Null hypothesis: *accuracy does not vary across runs.*

In other words, for a certain algorithm and dataset, we set out to measure *non-determinism.* To test this hypothesis, we use Levene's test [43] as follows: one sample contains the actual accuracy values for the 30 runs, the other sample has the

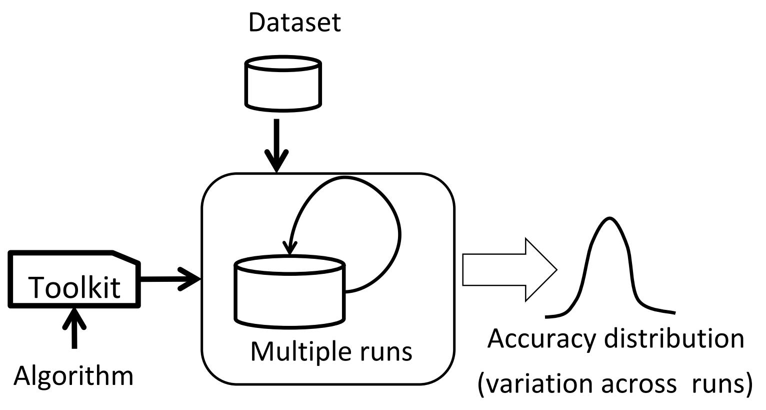

Figure 4.1 Testing for variation across runs.

same mean, size, and no variance, that is, all 30 elements are equal to the mean of the first set. We ran this on all datasets. Rejecting the null hypothesis means that accuracy varies in a statistically significant way across runs. We report results at $p < 0.05$.

Table 4.1 Levene's Test Results: the Number of Datasets, Out Of 162, With Significant Variance ($p < 0.05$)

Algorithm	**Toolkit**	**# Datasets**
kmeans++	sklearn	126
kmeans++	R	111
kmeans++	mlpack	144
kmeans++	matlab	125
kmeans++	shogun	143
kmeans++	tensorflow	144
kmeans++	weka	157
spectral	sklearn	93
spectral	sklearnfast	97
spectral	R	113
kmeans	sklearn	148
kmeans	R	153
kmeans	mlpack	146
kmeans	matlab	141
kmeans	shogun	146
kmeans	tensorflow	145
hierarchical	sklearn	71
hierarchical	R	63
hierarchical	matlab	63
gaussian	sklearn	136
gaussian	matlab	153
gaussian	tensorflow	151
gaussian	weka	123

Table 4.2 Top-10 Widest Differences in Accuracy Across Runs

Algorithm	**Toolkit**	**Dataset**	**Min**	**Max**
gaussian	tensorflow	prnn_crabs	-0.005	1
gaussian	matlab	prnn_crabs	-0.005	0.979
gaussian	matlab	analcatdata_cr.	-0.024	0.958
gaussian	tensorflow	twonorm	0	0.908
gaussian	matlab	twonorm	0.003	0.910
gaussian	tensorflow	ionosphere	0.004	0.772
spectral	R	breast-w	0.056	0.818
gaussian	matlab	analcatdata_aut.p	0.041	0.794
gaussian	matlab	wdbc	0.007	0.754
gaussian	tensorflow	breast-cancer-wsc.	0.032	0.760

In Table 4.1 we show the number of datasets where variance is statistically significant at $p < 0.05$; recall that we have a total of 162 datasets. We observe that Spectral is the most stable nondeterministic algorithm; for Spectral, only 93–113 datasets show significant variance. Hierarchical, which should be deterministic, still has 63–71 datasets with significant variance. In contrast, *K-means, K-means++, and Gaussian Mixture, have significant variance from run to run.*

In Table 4.2 we show how broad the accuracy range (difference between minimum accuracy and maximum accuracy) can be. The first three columns show the algorithm, toolkit and dataset. The last two columns show the minimum and maximum accuracy attained over the 30 runs. For example, Gaussian has quite a large range on some datasets: accuracy on the dataset prnn_crabs has a min-max range of *more than 1*, with one run's accuracy below 0 and another run having perfect or (close to perfect) accuracy.

In Table 4.3 we show how high the standard deviation of the accuracy can be across runs. For example, accuracy on the dataset twonorm can have a *standard*

Table 4.3 Highest Standard Deviations in Accuracy Across Runs

Algorithm	**Toolkit**	**Dataset**	**Stddev**
gaussian	tensorflow	twonorm	0.400
gaussian	tensorflow	prnn_crabs	0.345
gaussian	tensorflow	ionosphere	0.298
gaussian	sklearn	breast	0.281
kmeans++	weka	australian	0.236
gaussian	matlab	house-votes-84	0.236
gaussian	matlab	tokyo1	0.216
gaussian	sklearn.0_tol	breast	0.213
gaussian	matlab	twonorm	0.212
gaussian	matlab	analcatdata_cr.	0.206
gaussian	matlab	wine-recognition	0.205
spectral	R	appendicitis	0.204
kmeans++	shogun	house-votes-84	0.201

deviation of 0.4. More than a dozen other toolkit/algorithm setups have *standard deviation higher than 0.2*.

4.2 Variation Across Toolkits

In this section, we test a null hypothesis on how different implementations of the same algorithm in different toolkits lead to different clusterings. This testing procedure is shown in Figure 4.2: two toolkits implementing the same algorithms are run on the same dataset multiple times (30 in our case), and a statistical analysis is performed to compare the two accuracy distributions.

Null hypothesis: *For a given algorithm, accuracy does not vary across toolkits.*

To test this hypothesis, we use the Mann-Whitney U test as follows. We fix the algorithm, e.g., K-means++, and the dataset. Next, we compare the distributions of accuracy values pairwise, for all possible toolkits pairs, that is, if we have N toolkits for a given algorithm, for a given dataset there will be $\binom{N}{2}$ Mann-Whitney U tests;

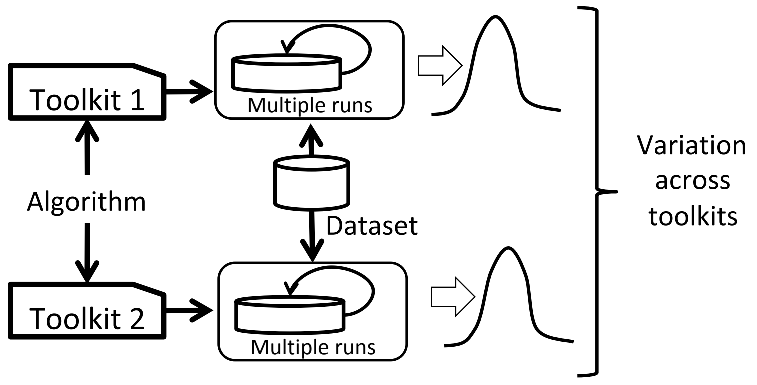

Figure 4.2 Testing for variation across toolkits.

hence for each algorithm there will be $162 \times \binom{N}{2}$ tests. Rejecting the null hypothesis means that accuracy varies significantly between toolkits. We report results at $p < 0.05$.

In Table 4.4, we show the number of datasets where accuracy distributions between two toolkits are statistically significant at $p < 0.05$. We observe that Gaussian Mixture and K-means++ induce most differences in toolkit outcomes' distributions (generally over 100 out of 162). Even for apcluster (deterministic), on 40 out of 162 datasets we found statistically significant differences between Sklearn and R.

4.2.1 Non-overlaps

In Table 4.5, we show the largest gaps between accuracy intervals, computed as follows: we find all dataset/algorithm combinations where the accuracy intervals for two toolkits, say $[ARI_{1min}, ARI_{1max}]$ and $[ARI_{2min}, ARI_{2max}]$ are non-overlapping, that is, $ARI_{1min} > ARI_{2max}$. In other words, *for any run of toolkit 1, its accuracy floor (min.) is higher than the accuracy ceiling (max.) of any run of toolkit 2.* We call that difference "gap", i.e., $gap = ARI_{1min} - ARI_{2max}$. We show the top-10 gaps in Table 4.5. Notice that this gap can be as large as 0.966.

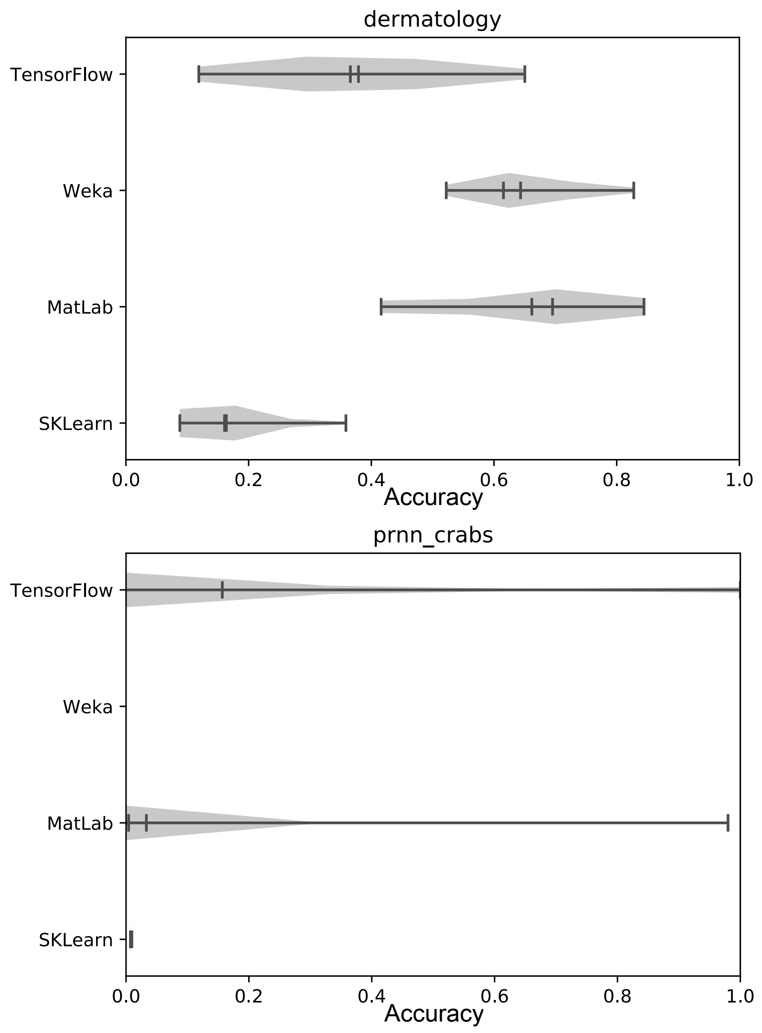

Figure 4.3 EM (Gaussian Mixture): differences between toolkits on two datasets, dermatology and prnn-crabs.

We found that 1,776 such gaps exist (out of 34,987 runs of the same algorithm/dataset combinations). This is very problematic, as it shows how toolkits are not "created equal" – even after multiple runs, in 1,776 scenarios, a toolkit's *best* accuracy cannot even reach another toolkit's *worst* accuracy.

In Figure 4.3, we show violin plots of toolkits' accuracy distributions in the EM algorithm on two datasets. On set dermatology (top) note the wide ranges of TensorFlow and the gap between WEKA and Sklearn. On set prnn-crabs (bottom) note the high-end accuracy of 1 (TensorFlow, MATLAB) and the consistently low accuracy in WEKA and Sklearn.

4.3 Variation Across Algorithms

In this section, we test a null hypothesis on the toolkit's impact when comparing algorithms. This testing procedure is shown in Figure 4.4: implementations of two different algorithms but in the same toolkit are run on the same dataset multiple times (30 in our case), and a statistical analysis is performed to compare the two accuracy distributions.

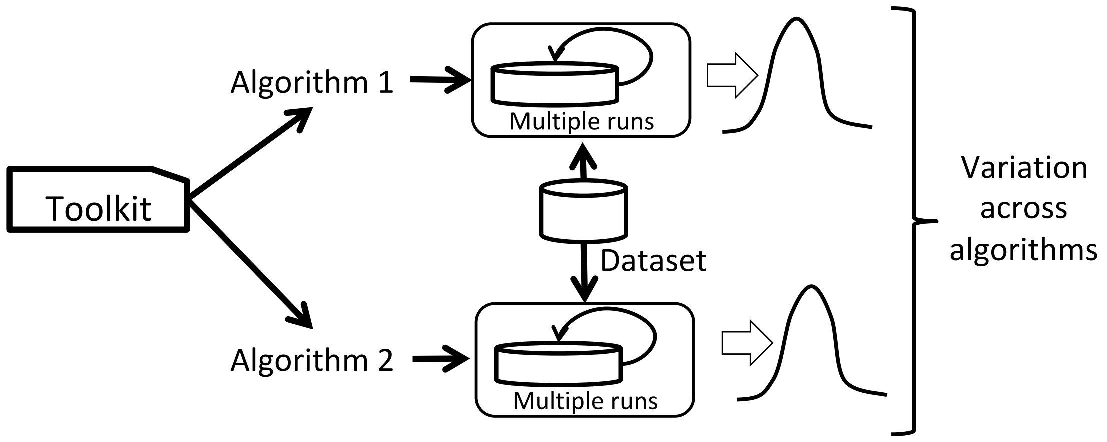

Figure 4.4 Testing for variation across algorithms.

Null hypothesis: *For a given toolkit, accuracy does not vary across algorithms.*

To test this hypothesis, we again use the Mann-Whitney U test. We fix the toolkit, e.g., MATLAB, and the dataset. Next, we compare the distributions of accuracy values pairwise, for all possible algorithm pairs. Rejecting the null hypothesis implies that, for a given toolkit, algorithms' accuracy varies significantly.

In Table 4.6, we show the number of datasets where accuracy distributions between two algorithms are significantly different. Typically, algorithms' accuracies differ on more than 110 datasets; we expected to see such differences between algorithms. However, we did not expect wide differences when looking at the *same* *algorithm pairs in different toolkits.* For example, K-means and K-means++ differ on 105/101/115/120 datasets in Sklearn/R/MATLAB/WEKA but only on 27 datasets in MLpack and only 31 datasets in Shogun. This again shows that toolkits are not interchangeable (though users might expect them to be).

4.4 Toolkit Disagreement

In this section, we test on a null hypothesis how different toolkits "disagree". We next set out to study whether toolkits "agree" or "disagree" on those points that are misclassified w.r.t. ground truth. Specifically, we are interested in those cases where two toolkits have relatively high accuracy w.r.t. ground truth, but there are large disagreements between the toolkits on the remaining, or misclassified points (i.e., where toolkits' clustering differs from ground truth).

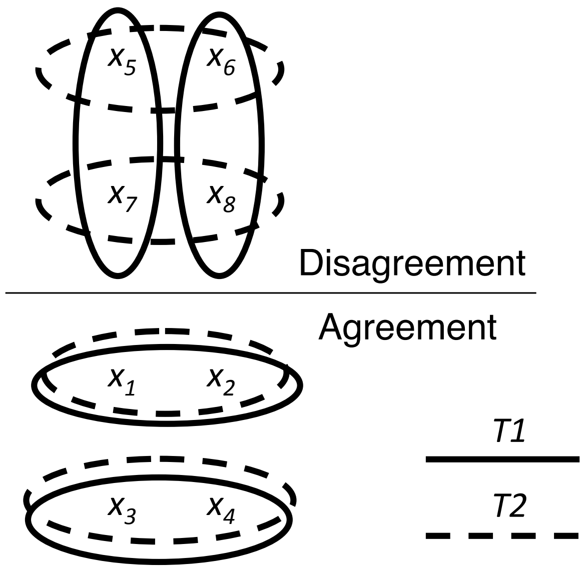

Figure 4.5 Toolkit disagreement.

We illustrate this in Figure 4.5. Assuming two toolkits $T1$ and $T2$, their clustering of x_1, x_2, x_3, x_4 (on the bottom) is in agreement, and let us assume this clustering agrees with ground truth as well. We want to measure the disagreement on the remaining points x_5, x_6, x_7, x_8 (on top).

Intuitively, datasets that induce this disagreement between $T1$ and $T2$ on the top points manage to expose differences in toolkit implementations "at the margin"; since agreement with ground truth is high, users might expect the toolkits will be in agreement on the reminaing points, too.

Let ARI_{T1G} and ARI_{T2G} be the accuracy of two different toolkits on the same algorithm and same dataset. Let ARI_{T1T2} be the ARI when comparing the two clusterings (rather than with ground truth). There were 14,831 ARI_{T1T2} comparisons. Out of these, we found 928 cases where:

$$ARI_{T1G} > ARI_{T1T2} \wedge ARI_{T2G} > ARI_{T1T2}$$

That is, there were *928 cases where toolkits' clusterings disagree with each other more than they disagree with ground truth* – in other words, toolkits disagree strongly on those points that are not clustered perfectly.

In Table 4.7, we show the top-10 such disagreements, excluding the trivial cases where one toolkit's accuracy is 1. These datasets are particularly important as they manage to "drive wedges" between toolkits; this has many applications, from differential toolkit testing to constructing adversarial datasets.

4.5 Conclusion

This chapter presented our approach for testing clustering implementations via rigorous statistical analysis, more specifically statistical analysis of clustering outcomes across multiple runs, toolkits, and algorithms. We found statistically significant variations across all these dimensions, which violate users' determinism, invariance, and consistency assumptions. Our results point out the need for improving determinism and consistency of clustering implementations. In Chapter 5 we show several root causes for nondeterminism and inconsistency. We also show that addressing those root causes improves determinism, increases consistency, and can even improve efficiency.

Table 4.4 Mann-Whitney U-test Results for Toolkits: Number of Datasets with Significantly Different Accuracy Distributions ($p < 0.05$)

Algorithm	**Toolkits**	**# Datasets**
kmeans++	sklearn vs. R	50
kmeans++	sklearn vs. matlab	107
kmeans++	sklearn vs. weka	134
kmeans++	sklearn vs. mlpack	104
kmeans++	sklearn vs. shogun	110
kmeans++	sklearn vs. tensorflow	109
kmeans++	R vs. matlab	104
kmeans++	R vs. weka	134
kmeans++	R vs. mlpack	101
kmeans++	R vs. shogun	108
kmeans++	R vs. tensorflow	115
kmeans++	matlab vs. weka	141
kmeans++	matlab vs. mlpack	105
kmeans++	matlab vs. shogun	120
kmeans++	matlab vs. tensorflow	124
kmeans++	weka vs. mlpack	96
kmeans++	weka vs. shogun	113
kmeans++	weka vs. tensorflow	125
kmeans++	mlpack vs. shogun	15
kmeans++	mlpack vs. tensorflow	57
kmeans++	shogun vs. tensorflow	60
spectral	sklearn vs. R	109
kmeans	sklearn vs. R	41
kmeans	sklearn vs. matlab	9
kmeans	sklearn vs. mlpack	8
kmeans	sklearn vs. shogun	9
kmeans	sklearn vs. tensorflow	9
kmeans	R vs. matlab	48
kmeans	R vs. mlpack	39
kmeans	R vs. shogun	43
kmeans	R vs. tensorflow	42
kmeans	matlab vs. mlpack	2
kmeans	matlab vs. shogun	1
kmeans	matlab vs. tensorflow	0
kmeans	mlpack vs. shogun	1
kmeans	mlpack vs. tensorflow	2
kmeans	shogun vs. tensorflow	1
hierarchical	sklearn vs. R	53
hierarchical	sklearn vs. matlab	58
hierarchical	R vs. matlab	57
gaussian	sklearn vs. matlab	129
gaussian	sklearn vs. weka	146
gaussian	sklearn vs. tensorflow	120
gaussian	matlab vs. weka	146
gaussian	matlab vs. tensorflow	104
gaussian	weka vs. tensorflow	120
dbscan	sklearn vs. R	0
dbscan	sklearn vs. mlpack	7
dbscan	R vs. mlpack	7
apcluster	sklearn vs. R	40

Table 4.5 Top-10 Largest Accuracy Gaps Between Toolkits

Algorithm	Dataset	Toolkit 1	Floor (Min)	Toolkit 2	Ceiling (Max)	Gap
gaussian	promoters	sklearn	1	tensorflow	0.034	0.966
gaussian	promoters	sklearn	1	tensorflow	0.034	0.966
spectral	promoters	R	0.962	sklearn	0.001	0.962
spectral	analcatdata_cred.	sklearn	0.84	R	-0.002	0.842
kpp	breast	weka	0.813	sklearn	-0.003	0.815
kpp	breast	weka	0.813	mlpack,matlab,R	0.02	0.792
kpp	breast	weka	0.813	tensorflow,shogun	0.02	0.792
gaussian	promoters	sklearn	1	matlab	0.234	0.766
kpp	promoters	tensorflow	1	weka	0.406	0.594

Table 4.6 Mann-Whitney U-test Results for Algorithms: Number of Datasets with Significantly Different Accuracy Distributions ($p < 0.05$)

Toolkit	**Algorithms**	**# Datasets**
sklearn	kmeans vs. kmeans++	105
sklearn	kmeans vs. gaussian	123
sklearn	kmeans vs. hierarchical	134
sklearn	kmeans vs. spectral	112
sklearn	kmeans vs. dbscan	150
sklearn	kmeans vs. apcluster	115
sklearn	kmeans++ vs. gaussian	132
sklearn	kmeans++ vs. hierarchical	153
sklearn	kmeans++ vs. spectral	109
sklearn	kmeans++ vs. dbscan	155
sklearn	kmeans++ vs. apcluster	117
sklearn	gaussian vs. hierarchical	145
sklearn	gaussian vs. spectral	108
sklearn	gaussian vs. dbscan	150
sklearn	gaussian vs. apcluster	113
sklearn	hierarchical vs. spectral	120
sklearn	hierarchical vs. dbscan	155
sklearn	hierarchical vs. apcluster	122
sklearn	spectral vs. dbscan	122
sklearn	spectral vs. apcluster	115
sklearn	dbscan vs. apcluster	117
shogun	kmeans vs. kmeans++	31
R	kmeans vs. kmeans++	101
R	kmeans vs. hierarchical	139
R	kmeans vs. spectral	94
R	kmeans vs. dbscan	149
R	kmeans vs. apcluster	117
R	kmeans++ vs. hierarchical	150
R	kmeans++ vs. spectral	97
R	kmeans++ vs. dbscan	157
R	kmeans++ vs. apcluster	123
R	hierarchical vs. spectral	113
R	hierarchical vs. dbscan	154
R	hierarchical vs. apcluster	123
R	spectral vs. dbscan	115
R	spectral vs. apcluster	122
R	dbscan vs. apcluster	123
tensorflow	kmeans vs. kmeans++	74
tensorflow	kmeans vs. gaussian	117
tensorflow	kmeans++ vs. gaussian	121
matlab	kmeans vs. kmeans++	115
matlab	kmeans vs. gaussian	135
matlab	kmeans vs. hierarchical	141
matlab	kmeans++ vs. gaussian	146
matlab	kmeans++ vs. hierarchical	155
matlab	gaussian vs. hierarchical	146
mlpack	kmeans vs. kmeans++	27
mlpack	kmeans vs. dbscan	135
mlpack	kmeans++ vs. dbscan	130
weka	kmeans++ vs. gaussian	120

Table 4.7 Top-10 Largest Disagreements Between Toolkits Yet Having High Agreement with Ground Truth

Algorithm	**Dataset**	**Toolkit1**		**Toolkit2**		
			ARI_{T1G}		ARI_{T2G}	ARI_{T1T2}
spectral	promoters	sklearnfast	0.889	R	0.962	0.853
gaussian	iris	weka	0.759	sklearn	0.904	0.693
gaussian	wine-recognition	weka	0.915	sklearn	0.607	0.568
gaussian	analcatdata_authorship	weka	0.951	sklearn	0.740	0.719
gaussian	wine-recognition	sklearn	0.607	matlab	0.724	0.469
spectral	breast-w	sklearnfast	0.809	R	0.552	0.477
gaussian	wine-recognition	weka	0.915	matlab	0.724	0.718
gaussian	iris	sklearn	0.904	matlab	0.560	0.550
gaussian	iris	tensorflow	0.562	sklearn	0.904	0.555
gaussian	texture	tensorflow	0.694	sklearn	0.742	0.614
gaussian	dermatology	weka	0.615	matlab	0.695	0.519
kpp	analcatdata_authorship	weka	0.777	shogun	0.718	0.700

CHAPTER 5

EXPOSING ROOT CAUSES OF IMPLEMENTATION-INDUCED INCONSISTENCY AND NONDETERMINISM IN DETERMINISTIC ALGORITHMS

In Chapter 4, we chose 7 popular clustering algorithms, 4 nondeterministic (K-means, K-means++, Spectral Clustering, Expectation Maximization-GaussianMixture); and 3 deterministic (Hierarchical clustering Agglomerative, Affinity Propagation, DBSCAN) and we analyzed clustering behavior on 162 datasets. We found statistically significant variations across runs, toolkits, and algorithms. In this chapter, we expose several root causes for nondeterminism and inconsistency and show that remedying these root causes improves determinism, increases consistency, and can even improve efficiency. We use a substantially higher number of datasets (528 vs. the 162 used previously), strengthening the relevance of our statistical findings.

5.1 Definitions and Experimental Setup

We use SmokeOut's suite of differential clusterings to measure determinism and inconsistency for implementations of deterministic clustering algorithms.

5.1.1 Datasets

We used 528 datasets from OpenML [11]. 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 [44], a benchmark suite specifically designed to evaluate ML implementations. Table 5.1 summarizes the characteristics of our datasets: on average, datasets have 454 instances, 39 dimensions, and 2.6 clusters.

Table 5.1 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

5.1.2 Algorithms and Toolkits

We studied three deterministic algorithms (Affinity Propagation, DBSCAN, Hierarchical Agglomerative Clustering) and the deterministic part of K-means (Section 2.5). We examined several toolkits: MATLAB, MLpack, R, Scikit-learn, and TensorFlow (Section 2.4).

5.2 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.

5.2.1 Inconsistency

We measure inconsistency using mutual ARI (defined in Section 2.2). 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 5.2 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

Table 5.2 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

inconsistency root causes and shows how addressing these root causes is effective at reducing inconsistency (the remaining Table 5.2 rows are explained in Sections 5.2.4 and 5.2.5).

5.2.2 Case Study 1: Bounding the Number of Iterations

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

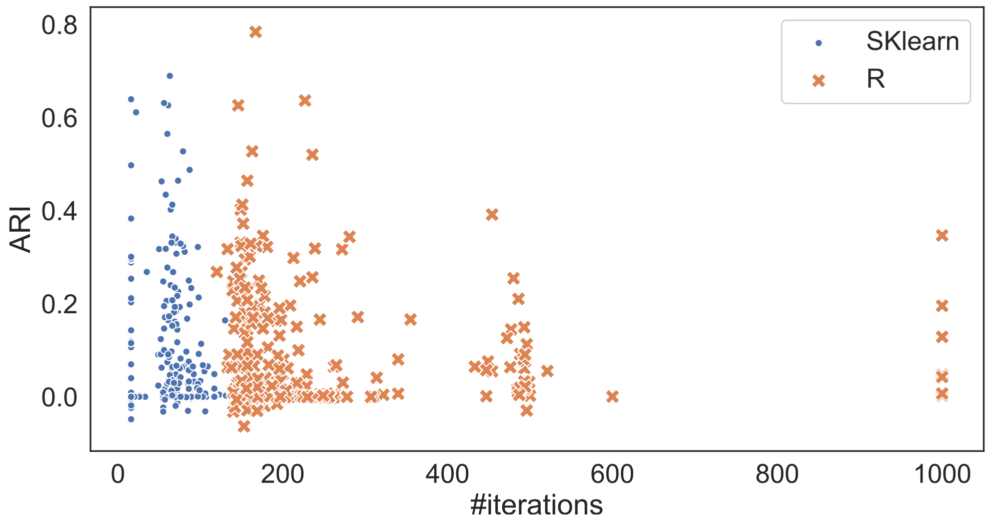

Figure 5.1 Affinity Propagation's accuracy vs. #iterations in Scikit-learn and R.

We illustrate this in Figure 5.1 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 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 ($p-$value > 0.1). However, a paired test on #iterations until stopping (i in Figure 2.5) shows significant differences ($p-$value ≈ 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).

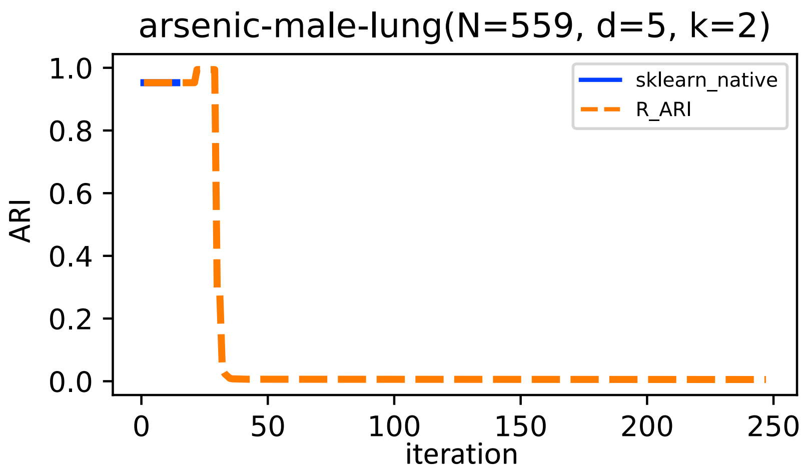

Figure 5.2 Lose-lose due to over-iterating in R (orange dashed line); note the higher #iterations and lower final accuracy compared to Scikit-learn (blue).

Finally, note the "hard" limits for MAX_ITER at 200 and 1000, respectively – the 1000 vertical line is clearly visible for R in Figure 5.1 – 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.

5.2.3 Under-iterating and Over-iterating

Figure 5.2 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.

Conversely, Figure 5.3 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 5.3 shows the highest margins for Scikit-learn and R, respectively. The first column contains the dataset name, the next four columns show the iterations

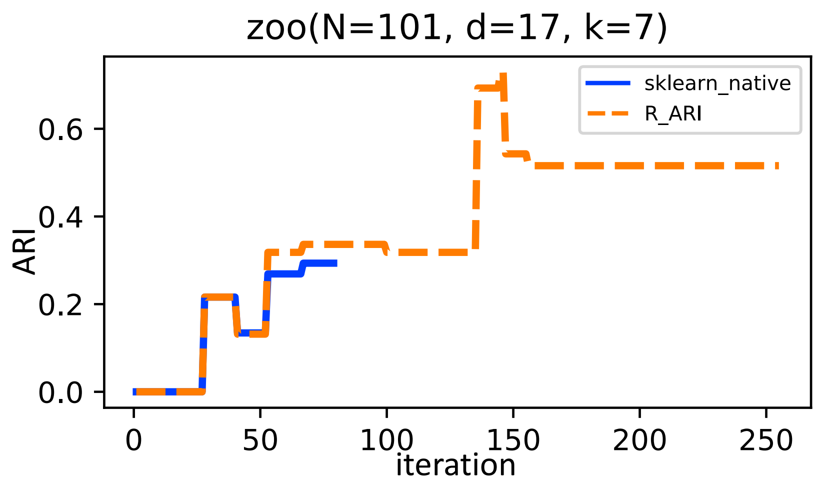

Figure 5.3 Under-iterating – premature termination – leads to lower accuracy in Scikit-learn (blue solid line) compared to R (orange dashed line).

Table 5.3 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

Table 5.4 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

and accuracy for Scikit-learn and R, respectively, while the last column shows the accuracy difference (absolute value).

Note how, on the arsenic-* datasets,[1] 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.

5.2.4 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 ($p-$value $= 0.16$). Second, the high-margin discrepancies between the two toolkits were removed or reduced substantially. In Table 5.4 we show the largest accuracy gaps after implementing this control. Note that accuracy differences were at most 0.17 (in stark contrast with Table 5.3 where

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

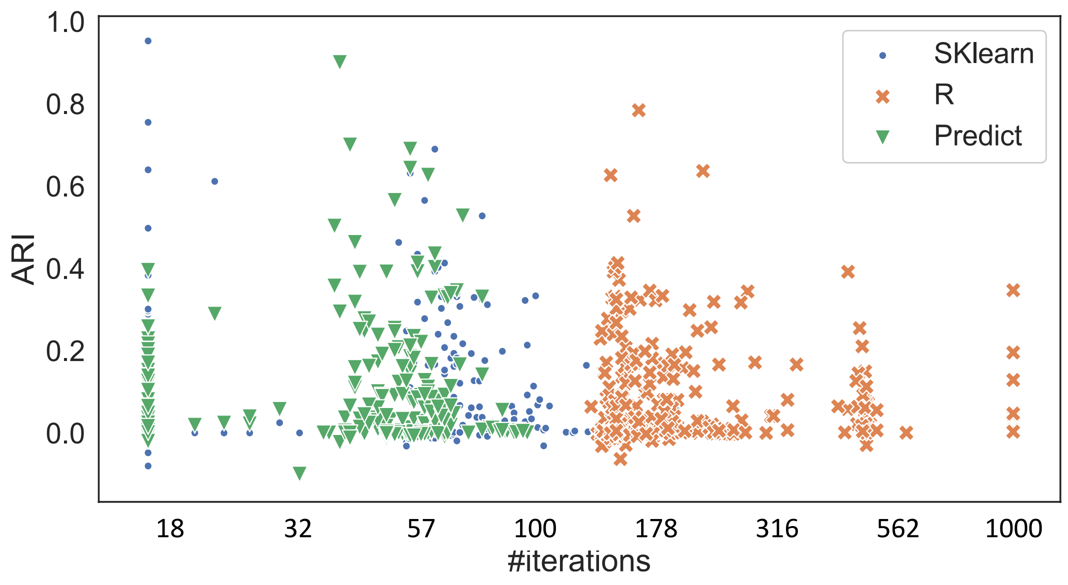

Figure 5.4 ARI vs. #iterations: Scikit-learn predicted (green crosses), Scikit-learn default (blue triangles), R default (orange circles); for legibility, x-axis is logarithmic.

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 5.2 show consistency improving from 0.68 (default) to 0.94 or 0.95, respectively, which indicate this is an effective control.

5.2.5 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: $R^2 = 0.883$, $t-$value $= 42$,

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

Figure 5.5 Noise insertion code.

$p-$value ≈ 0. For R, the regression did not find a good fit (Section 5.2.6 explains why).

Therefore, we constructed a model where MAX_ITER was predicted by $log(N)$. Figure 5.4 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 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 ($p-$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 5.2 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.

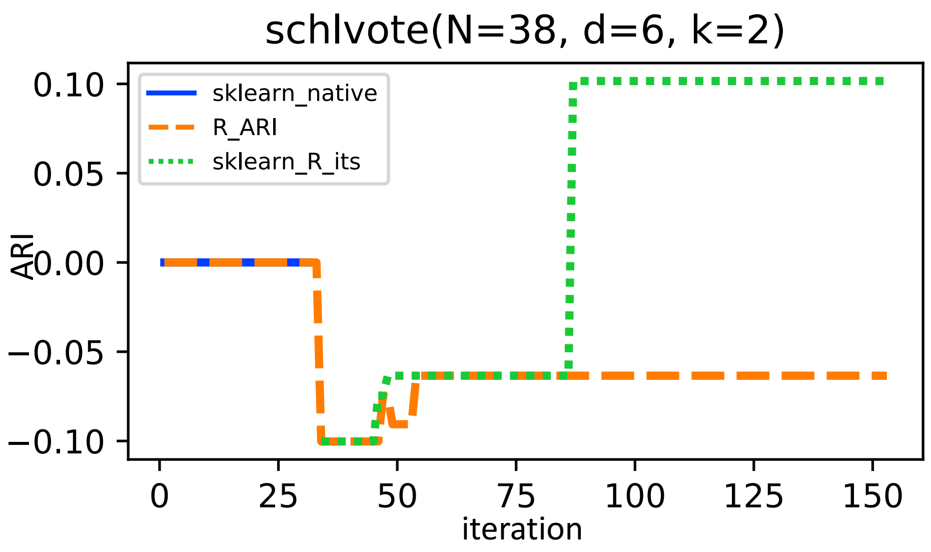

Figure 5.6 Differences due to noise, after controlling for #iterations: by default, Scikit-learn would terminate quickly and at low accuracy (blue). Forcing Scikit-learn to keep iterating improves accuracy (green, dotted line). R's accuracy shown in orange dashed line.

5.2.6 Noise

Another source of inconsistency we discovered was the noise insertion policy. Essentially, toolkits choose to add "noise" to prevent degenerate clustering scenarios. Figure 5.5 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 $1e-15*s$ 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.

Inconsistency. Figure 5.6 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 ($p-$value < 0.05).

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

Table 5.5 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 5.6 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 5.5 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 5.6 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.

5.2.7 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.

5.3 DBSCAN

DBSCAN forms clusters by looking for "dense" regions, i.e., regions with at least *minPoints* separated by a maximum distance *eps*. 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.

Table 5.7 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**
$\boxed{\text{Defaults}}$	*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	
$\boxed{\text{Controlled } eps}$	*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	
$\boxed{\text{Heuristic } minPts}$	*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 5.8 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

5.3.1 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.

5.3.2 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 5.7: the top-5 datasets, with the largest gaps between toolkits. The difference between MLpack and the other toolkits across all datasets was marginal, $p-$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.

5.3.3 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 5.7: 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 $p-$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.

5.3.4 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" [30]. Therefore, we set *minPts=d+1*, where d is the dimensionality of the dataset; we present the results in the bottom rows of Table 5.7. The difference between MLpack and the other toolkits across all datasets was significant, $p-$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.

5.3.5 Mutual ARI

The measurements so far have used accuracy (ARI vs. Ground Truth). Table 5.8 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 2.2). 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).

5.3.6 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*.

Table 5.9 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	

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.

5.4 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).

5.4.1 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.

Table 5.10 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

5.4.2 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 5.9, 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 [50]: 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 5.9. 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 [10]). Per its authors [12], 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.

Mutual ARI. The mutual ARIs are presented in Table 5.10. 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 5.10) the mean mutual ARI has increased to 0.93–0.94.

5.4.3 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.

5.5 K-means

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.

Table 5.11 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

5.5.1 Inconsistency

We show the progression toward stronger consistency, starting from default parameters and then applying stronger controls: Table 5.11 shows the number of datasets that toolkits disagree on, while Table 5.12 shows the mean mutual ARIs and the strongest disagreements.

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.[2]

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 5.11 shows that R's disagreement with the other toolkits reduces substantially, from 369–370 disagreements to just 15–21; Table 5.12 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,

[2]R uses "Hartigan-Wong" heuristics [32] by default, whereas Scikit-learn and MATLAB use "Lloyd" heuristics [39].

Table 5.12 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

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 5.11.

"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 5.12. Note that mutual ARIs are at least 0.99.

Table 5.11 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.

5.5.2 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.

5.6 Conclusion

In this chapter we were able to manually identify, and expose, various root causes of nondeterminism or inconsistency in implementations of deterministic algorithms: default parameter settings, noise insertion, distance metrics, or termination criteria. Controlling these sources can eliminate nondeterminism and bring several different implementations of the same algorithm more in line with each other. In Chapter 6 we address the challenge associated with manually finding root causes by exposing root causes *automatically*.

CHAPTER 6

AUTOMATIC DETECTION OF NONDETERMINISM AND INCONSISTENCY ROOT CAUSES

In Chapter 5, we (a) exposed various root causes of nondeterminism or inconsistency – default parameter settings, noise insertion, distance metrics, termination criteria – and (b) eliminated them to improve effectiveness and efficiency. To automatically find inconsistencies across multiple clustering implementations, in this chapter we introduce a programmer-assisted approach to trace and compare clustering implementations. Our approach uses annotations, coupled with dynamic analysis, to trace programs and automatically find nondeterminism/inconsistency root causes. We evaluate our approach on Scikit-learn, R, and Elki. Our results show that all examined toolkits can be transformed with modest programmer effort; and that our approach is effective at automatically finding nondeterminism/inconsistency root causes.

6.1 Motivation

As prior chapters have shown, we manually discovered root causes of nondeterminism and inconsistency of in deterministic clustering algorithm's implementations for four algorithms. However, finding root causes automatically is still a challenge for two main reasons: when clustering algorithms vary (e.g., K-means with different heuristics) and when implementations' programming languages vary (e.g., clustering implemented in Python, R, or Java).

We start by motivating our approach with clustering accuracy in `tokyo1` on AP. `tokoyo1` contains performance co-pilot (PCP) data for the Tokyo server at Silicon Graphics International (SGI). We compared clustering outcomes of deterministic algorithm AP in two toolkit configurations. In Figure 6.1, we can find mutual ARIs

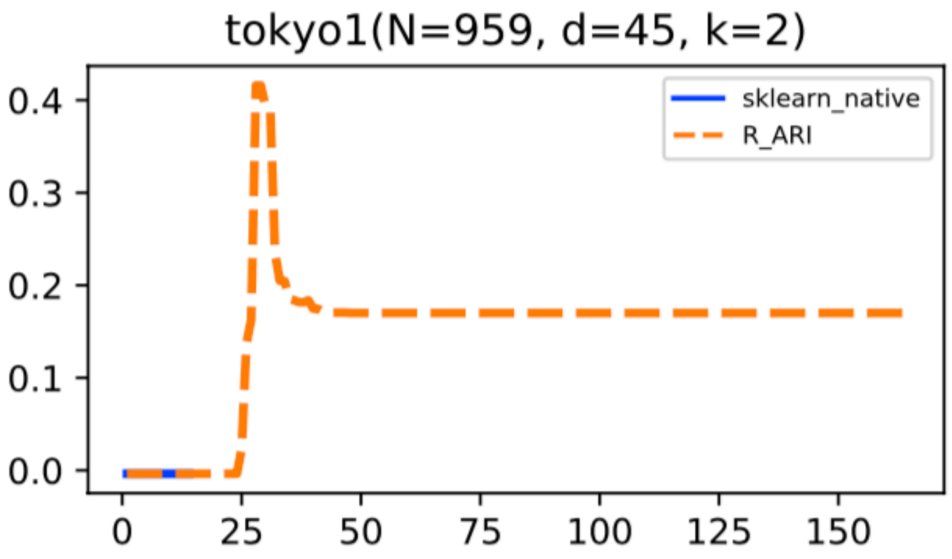

Figure 6.1 AP: Accuracy distribution for sckit-learn and R on dataset `tokyo1`.

for each iteration which indicates inconsistencies between Scikit-learn and R due to under-iterating. The question is, what specific part of the implementation (code, setting, input parameter) is responsible for these inconsistencies?

To answer this question, we introduce a programmer-assisted approach where programmers simply indicate algorithm *phases* and our toolchain will isolate and identify the inconsistency (or nondeterminism) root causes.

In Section 3.1, we describe the architecture of our approach. Our approach supports either compile-time or run-time solutions (annotation frameworks) for transforming clustering programs into programs that output "differentiable" traces, a dynamic analysis that performs the differencing, and finally an automatic approach to find root causes on trace reports.

Constructing effective annotation frameworks is key, with reducing annotation burden being a top concern. Therefore, in our approach, described in Section 6.3, programmers simply add a few annotations to indicate certain variables and/or algorithm phases, and writing conversion functions.

In Section 6.4, we provide an evaluation of approach from two perspectives: effectiveness and ease of use. We evaluated our approach on three clustering algorithms: AP, K-means and DBSCAN. We found that programming effort is overall modest, and that most of the effort consists of writing conversion functions.

6.2 Overview

Figure 6.2 shows the architecture for our approach. There are three steps. First, we apply an annotation framework to transform clustering programs so their phases can be traced and we use a dynamic analysis to trace the deterministic part of the implementation. Last, we use a differential analysis to detect inconsistency on trace reports and automatically find root causes.

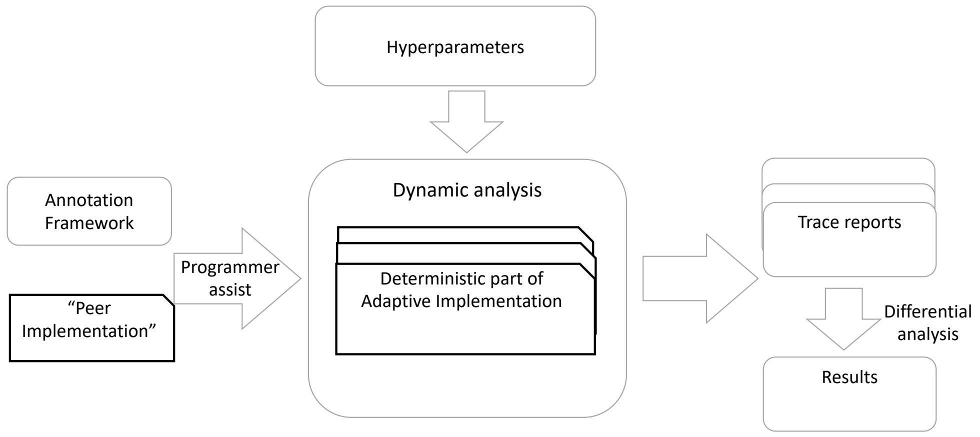

Figure 6.2 Architecture.

Annotation Framework. We describe our approach for transforming the clustering program into adaptive implementation in Section 6.3.

Dynamic Tracing. Our dynamic tools trace program executions to identify program phases and make these phases available in the dynamic trace. We now discuss the language-specific tracing approach.

For Python, a Python-execution tool [2] traces the local context of a Python function's execution; we modified it to trace multiple functions.

For Java, we use a runtime dynamic tracing tool called Java Tracing Agent [3]. This is a lightweight and fast runtime injection tool for logging and tracing.

For R, note that R's `trace` library only allows tracing the entry and exit points of R closures. Therefore we used the RDT dynamic tracing framework 10.1145/3360579 that modifies GNU R Virtual Machine with probes that can trace function entry and exit, variable definition and mutation, non-local jumps and S3/S4 dispatch, etc. These probes are triggered when specific program events are called.

To ensure our results are deterministic, we turn off implementation-induced randomness before applying dynamic analysis.

Inconsistency Detection. We apply a differential analysis on the trace reports. These trace reports from the adaptive programs are implementation-agnostic. Our comparisons have two components: initial variables in the initial and iterative variables in the iteration phase. There are two main reasons for this design. First, initial variables are traced only once, but iterative variables are updated for each iteration. Second, our evaluation shows that most inconsistencies are caused by inconsistent implementation inputs.

6.3 Annotation Framework

We now present our annotation framework that transforms a general clustering implementation into a trace-emitting differentiable implementation. Programmers need to add a handful of annotations to the source code, to indicate the traced functions; mark the initial phase and iteration phase; and write convention functions that switch program to be adaptive. This annotated source code is identified and traced by our modified dynamic tracing tools.

Running example: Annotation Framework on Python for Affinity Propagation.

In Figure 6.3, we show an excerpt from Affinity Propagation (AP) implemented in Python – the main function implementing AP algorithm. The inputs of this function are similarity matrix, preference, convergence_iter, max_iter; they are all initial variables. The iterative variables are it, I, E, K. However we only trace K, since K is used to identify exemplars and affect results directly. To transform the program to adapt to its inputs and algorithm, we add two conversion functions: _initial_variables and _iteration_variables .

Running example: Conversion Functions on Java for Affinity Propagation.

In Figure 6.4, we show an excerpt from Affinity Propagation implemented in Java (the implementation's core method). The inputs to the method are similarity matrix, lambda, convergence_iter, max_iter. There are slight differences from Python: the variable lambda is the damping factor in Python; the preference is the first diagonal value of similarity matrix according to the source code. The iterative variables are it, K, etc., and we still pick up K, because K is used to identify exemplars. To transform the program we insert two conversion functions: initialPhase and iterationPhase. Since the dynamic tracing tool on Java will pass tracing configures on runtime, programs can perform switching by passing settings for conversion functions on configuration and there is no need to add annotations on program again.

Programming Model. Our approach is designed to minimize the programmer's burden. Programmers simply use two annotations to indicate initial variables and variables in the loop. Also, programmers need to write conversion functions, although tracing these functions is automatic. Our modified tracing tools will identify these functions in program executions and restore their data.

Table 6.1 Annotations on Clustering implementations.

	Annotation	***Modify Implementation***	***Purpose***
Python	**record_initial**	Yes	input
Python	record_iteration	Yes	loop
Java	initialPhase	Yes	input
Java	iterationPhase	Yes	loop
R	initial_phase	No	adaptive input & restore data
R	iteration_phase	No	adaptive loop & restore data

Note that the insertion place (where programmers add annotations) differs depending on the underlying language. For example, while Python programmers should add annotations **record_initial** and record_iteration in the source code, Java programmers indicate conversion functions initialPhase and iterationPhase into Java dynamic tools. Because R is recompiled to insert probes, after inserting annotations in R, the R project itself needs to be recompiled.

6.4 Evaluation

We evaluate our annotation framework on three clustering algorithms: Affinity Propagation, K-means and DBSCAN as implemented in scikit-learn, R (apcluster package version 1.4.7, stats version 3.2.3 package) and Elki. Our datasets come from OpenML (Section 5.1.1).

6.4.1 Effectiveness

Affinity Propagation. To avoid testing on nondeterministic part of implementation, we turn off the random generator and remove noise from the similarity matrix. Note that Elki does not use noise insertion. We measure inconsistency using mutual ARIs for clustering. We analyze on trace reports to find inconsistent variables that indicate root causes.

Table 6.2 shows that the number of inconsistent datasets found by our tools. 341 out of 446 datasets are inconsistent because of inconsistent initial variables of max_iter and conv_iter between scikit-learn and R implementation. Also, Elki has 22 inconsistent datasets with the other two toolkits.

We also found a substantial number of inconsistent datasets for Elki: 364 out of 446 datasets are inconsistent when Elki is compared with scikit-learn and R. The root cause is the function Elki uses, a median function in the method QuickSelect.quantile.

These results indicate that our tool is effective: it shows that inconsistent initial variables are root causes of inconsistencies. Note that we found only 6 inconsistent datasets in scikit-learn and R, caused by the iterative variable K.

Table 6.2 Number of Inconsistent Datasets for Affinity Propagation.

	Parameter	**R & Scikit-learn**	**Java & Scikit-learn**	**Java & R**	**Root Cause**
Initial Phase	preference	20	364	364	Inconsistent function
preference, damping	damping	0	0	0	None
max_iter & conv_iter	max_iter & conv_iter	341	22	22	Parameter settings
Iteration Phase, K	K	6	0	0	Unequal assignment

K-means. To avoid use random starting points, we feed the same initial centroids for all implementations scikit-learn, R, and Elki.

K-means has multiple heuristic algorithms, and to make implementations comparable, we compare K-means implementations of the same heuristic algorithm. We compare two heuristics: Lloyd in scikit-learn, R and Elki; Macqueen inn R and Elki. We trace centroids at initial phase and iteration phase.

Table 6.3 shows that the major root cause for inconsistency is the inconsistent implementations for K-means heuristic algorithms. For the Lloyd algorithm, there are no inconsistent cases found in the initial phase. All inconsistent cases are due to inconsistent centroids in the iteration process: 48 inconsistent datasets are found when Java (Elki) is compared to R; 49 inconsistent datasets are found when Java (Elki) is compared to scikit-learn. We also found that there are more inconsistent datasets for the Macqueen algorithm due to inconsistent centroids in the iteration phase than those on the initial phase. In total, there are 222 inconsistent datasets in the iteration phase. In conclusion, most of the inconsistent datasets using the Macqueen heuristic are in the iteration phase, and Macqueen is also easily affected by "bad" starting points compared to Lloyd heuristic (since we observed 59 inconsistent cases happening in the initial phase).

Table 6.3 Number of Inconsistent Datasets for K-means.

	Parameter	**Java & R(Lloyd)**	**Python & Java(Lloyd)**	**Java & R(MacQueen)**	**Root Cause**
Initial Phase, centroids	centroids	0	0	59	Empty Cluster
Iteration Phase, centroids	centroids	48	49	222	Unequal Assignments

DBSCAN. We studied this algorithm for three toolkits; we only compare scikit-learn/R with Elki, since there is no inconsistency observed between scikit-learn and R. Recall that DBSCAN forms clusters by looking for "dense" regions, i.e., regions with at least *minPoints* separated by a maximum distance *eps*. We only traced the initial variables; 2 out of 382 datasets were inconsistent.

Table 6.4 Program Size and Program Effort.

	Algorithm	**Size LOC**	**Step1 LOC**	**Step2 LOC**
Python	AP	484	4	4
	K-means	2017	4	4
	DBSCAN	309	4	2
Java	AP	386	2	2
	K-means	316	2	2
	DBSCAN	357	2	1
R	AP	432	9	2
	K-means	784	6	2
	DBSCAN	273	3	1

6.4.2 Manual Annotation Effort

Programming effort. Converting an off-the-shelf implementation into a differentiable implementation is a two-step process (hence we only focus on the programming effort for adding support for annotation framework): first, writing the conversion functions for initial phase and iteration phase; and second, annotating the source code.

We report this effort in Table 6.4. The function code size "Step 1" column depends on the number of functions and parameters. Python and Java need two conversion functions; R need two conversion functions with lines for restoring data.

"Step 2" code consists of annotations and the lines that support for annotations. For annotations (the four grouped columns), Python need four more lines because it needs adding two annotations and call for conversion functions. Java and R do not have to add the line for annotations.

6.5 Conclusion

We proposed a programmer-assisted approach to detect implementation-induced inconsistency for clustering implementations automatically. Our evaluation shows the effectiveness of our approach at automatically identifying root causes in three clustering implementations.

```
@record_initial(200, " initial .json")
def _initial_variables( preference , damping, convergence_iter , max_iter):
    pass


@record_iteration(200, " iteration .json")
def _iteration_variables(K, labels , it ):
    pass



def affinity_propagation (S, preference =None, convergence_iter=15, max_iter=200,
                         damping=0.5, copy=True, verbose=False,
                          return_n_iter =False):
S = as_float_array (S, copy=copy)
preference = np.median(S)
S. flat [::( n_samples + 1)] = preference
......
 _initial_variables ( preference , damping, convergence_iter , max_iter)
for it in range(max_iter):
     ......
    K = np.sum(E, axis=0)
     _iteration_variables (K, it )
```

Figure 6.3 Excerpt from Python Affinity Propagation; functions _initial_variables and _iteration_variables with annotations are inserted.

```
public void initialPhase(double lambda, double preference , int convergenceiter ,
    int maxiter);


public void iterationPhase( int iteration , int k);


public Clustering <MedoidModel> run(Relation<O> relation) {
  ArrayDBIDs ids = DBIDUtil.ensureArray(relation .getDBIDs());
  final int size = ids. size ();


  int [] assignment = new int[ size ];
  double [][] s = initialization . getSimilarityMatrix ( relation , ids );
  double [][] r = new double[size ][ size ], a = new double[size ][ size ];


  initialPhase (lambda, s [0][0] ,convergence, maxiter);
  int inactive = 0;
  for ( int iteration = 0; iteration < maxiter && inactive < convergence;
      iteration ++) {
  ...
  iterationPhase ( iteration , k);


 ...
}
```

Figure 6.4 Excerpt from Java Affinity Propagation; functions initialPhase and iterationPhase.

CHAPTER 7

RELATED WORK

We now compare our approach with prior efforts.

7.1 Testings on Clustering Implementations

While clustering is a richly explored field, prior clustering research efforts have not questioned or investigated clustering reliability or correctness. For example, Software Engineering research has used clustering as a tool rather than as an object of study; Machine Learning and Data Mining research can be split into theoretical research on clustering properties, or experiments on improving clustering; in both cases, the research literature assumes correct algorithm implementations.

The study closest to our approach in breadth of algorithm/toolkit combinations is Kriegel et al.'s [37]. They have also pointed out the peril of assuming that "toolkits don't matter": an algorithm's implementation is *not* standardized across all toolkits. They have compared several algorithms and implementations on a narrower benchmark set (a single dataset of 500k Twitter locations, and subsets thereof) but their goal was different: runtime efficiency. They found orders-of-magnitude differences across toolkits for the same algorithm and same input dataset.

Abu [14] has compared four clustering algorithms – K-means, hierarchical, SOM, and Expectation Maximization (EM), each implemented in two toolkits LNKnet and Cluster/TreeView; they used a single 600-instance dataset, and compared performance/accuracy on this dataset, and a 200-instance subset thereof. Our setup is substantially larger and our focus is substantially broader.

7.2 Machine Learning Research on Clustering Properties

Ben-Hur et al. [16] have investigated hierarchical clustering on several datasets: varying K to find the value for which the algorithm is most stable. Our goal is toolkit dependability, and our focus is on datasets with ground truth and fixed K.

Fred [27] has proposed voting K-means, an improvement upon standard K-means by choosing clusters on a majority voting policy, to weed out outliers. They use consistency (similarity of partitionings for multiple runs of K-means on the same dataset and the same K) which is akin to our notion of determinism. Their experiments were run with varying K on two datasets. Our use of ARI is more robust, and our goal is toolkit dependability, rather than improving K-means.

Fränti [26] has compared performance on clustering basic benchmark, and measure performance on four factors: overlap of clusters, number of clusters, dimensionality and unbalance of cluster sizes. However, they only consider synthesis data and their datasets have simple structures. Our work is based on PMLB which includes mainly real-world datasets allowed for comparing ML methods comprehensively.

Hamerly [31] has proposed a new algorithm for accelerating K-means, and performed an evaluation on efficiency similar to Kriegel et al.'s (time and memory). Our focus is on accuracy rather than efficiency.

Chen et al. [19] have compared four clustering algorithms – hierarchical clustering, K-means, Self-organizing Map (SOM) and Partitioning around Medoids (PAM) on a single dataset, mouse genomic data. Unlike us, they varied the K, whereas we used the ground truth's K. Our focus is different: varying runs of the same algorithm, and a breadth of datasets.

Clustering stability has been defined by Tilman et al. [38] as "solutions [that] are similar for two different data sets that have been generated by the same (probabilistic) source". Our definition of stability is different: similarity of solutions on the same dataset, but produced by two different runs.

7.3 Testing on Machine Learning Implementations

There are several research efforts on automatic testing of ML libraries.

Srisakaokul et al. [48] detect inconsistencies across multiple implementations of ML algorithms like kNN or Naive Bayes (NB). They use majority votes to estimate the expected output, but they assume that most algorithms are correctly implemented. In contrast, our approach found implementation-induced inconsistencies.

Dwarakanath et al. [24] apply transformations on the training and testing data to detect inconsistencies on ML libraries. However, we used benchmark datasets and would like to see how implementation-induced inconsistencies and nondeterminism affect our results in the real world.

7.4 Research on Improving Neural Networks Safety

Prior efforts have focused on supervised learning approaches, mostly Neural Networks (NNs)/Deep Learning [29], rather than unsupervised learning (e.g., clustering). While NNs are popular and successful, challenges such as limited training data, unlabeled data, or interpretability make "traditional" clustering preferable.

CHAPTER 8

FUTURE WORK

In this dissertation, we have proposed several techniques to expose nondeterminism and inconsistency on clustering implementations. However, there are several avenues worth exploring. In this section, we lay out some possible directions for future work.

8.1 Automatic Generation for Bug-Induced Datasets

In the prior study, we used datasets that had ground truth. Compared with ground truth, we can point out a“bad” sample that produces bugs and leads to wrong results. There are several studies about generating/transforming bug-induced datasets: Dutta et al. [23] use fuzzing to test probabilistic programming; Dwarakanath et al. [24] apply transformations on datasets and can find artificially injected bugs. Our future work can be extended to apply the technique of applying Bug-Induced datasets on implementations to find bugs in the search-based engineering area.

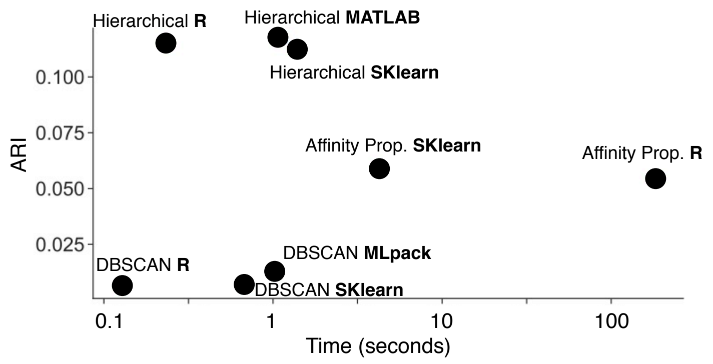

Figure 8.1 Mean ARI (linear scale) vs. mean time (log scale).

8.2 Improving Runtime Performance

Performance (clustering running time) is another strong differentiating factor between toolkits: in Figure 8.1, note the orders-of-magnitude difference in time between toolkits implementing the same algorithm. Therefore, techniques for *profiling* clustering implementations can expose runtime inefficiencies and eventually improve runtime performance.

CHAPTER 9

CONCLUSION

In summary, this dissertation makes the following contributions:

- This dissertation is the first approach to question and investigates clustering reliability or correctness.
- This dissertation has designed and implemented the SmokeOut tool to quantify clustering outcomes across different algorithms, toolkits, and multiple runs.
- This dissertation proposes a statically rigorous approach for comparing runs, toolkits and algorithms.
- This dissertation quantifies implementation-induced nondeterminism and inconsistency, finding their root causes, and showing how they can be alleviated.
- This dissertation introduced an approach to dynamically collect traces for clustering implementations and automatically find root causes of inconsistency/nondeterminism.

REFERENCES

[1] Companies using tensorflow. https://www.tensorflow.org/.

[2] execution-trace. https://github.com/mihneadb/python-execution-trace. Accessed: 2020-10-15.

[3] java-tracing-agent. https://github.com/dakaraphi/java-tracing-agent. Accessed: 2020-10-15.

[4] Tensorflow github. https://github.com/tensorflow/tensorflow.

[5] *SODA '07: Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, USA, 2007. Society for Industrial and Applied Mathematics.

[6] Cran task view: Cluster analysis & finite mixture models, November 2018. https://cran.r-project.org/web/views/Cluster.html.

[7] Matlab file exchange:clustering, November 2018. https://www.mathworks.com/matlabcentral/fileexchange/?term=clustering+product

[8] Weka mailing list, September 2018. http://weka.8497.n7.nabble.com/Weka-clustering-diverges-from-other-toolkits-td43955.html.

[9] Mathworks fast facts, April 2019. https://www.mathworks.com/company/aboutus.html.

[10] Nearest-neighbor chain algorithm, April 2019. https://en.wikipedia.org/wiki/Nearest-neighbor$_chain_algorithm.$

[11] OpenML, April 2019. https://www.openml.org/.

[12] The R Project: Hierarchical Clustering, April 2019. https://svn.r-project.org/R/trunk/src/library/stats/R/hclust.R.

[13] Who is using scikit-learn?, April 2019. http://scikit-learn.org/stable/testimonials/testimonials.html.

[14] Osama Abu Abbas. Comparison between data clustering algorithm. *Int. Arab Journal of Information Technology*, 5(3), 2008.

[15] Duncan Bell. 5 great ai-powered home devices that will improve your life today. https://www.t3.com/features/5-great-ai-powered-home-devices-that-will-improve-your-life-today.

[16] Asa Ben-Hur, André Elisseeff, and Isabelle 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*, pages 6–17, 2002.

[17] Guy Brys, Mia Hubert, and Anja Struyf. A comparison of some new measures of skewness. In Rudolf Dutter, Peter Filzmoser, Ursula Gather, and Peter J. Rousseeuw, editors, *Developments in Robust Statistics*, pages 98–113, Heidelberg, 2003. Physica-Verlag HD.

[18] Carlton E. Sapp. Gartner: Preparing and architecting for machine learning, 2017. https://www.gartner.com/binaries/content/assets/events/keywords/catalyst/catus8/preparing_and_architecting_for_machine_learning.pdf.

[19] Gengxin Chen, Saied A. Jaradat, Nila Banerjee, Tetsuya S. Tanaka, Minoru S. H. Ko, and Michael Q. Zhang. Evaluation and comparison of clustering algorithms in analyzing es cell gene expression data. *Stat. Sinica*, pages 241–262, 2002.

[20] Arthur P. Dempster, Nan M. Laird, and Donald B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[21] Nathalia Moraes do Nascimento, Carlos Lucena, Paulo S. C. Alencar, and Donald D. Cowan. Software engineers vs. machine learning algorithms: An empirical study assessing performance and reuse tasks. *CoRR*, abs/1802.01096, 2018.

[22] Harold E. Driver and Alfred Louis Kroeber. *Quantitative Expression of Cultural Relationships*. Publications in American archaeology and ethnology. University of California Press, 1932.

[23] Saikat Dutta, Owolabi Legunsen, Zixin Huang, and Sasa Misailovic. Testing probabilistic programming systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, page 574–586, New York, NY, USA, 2018. Association for Computing Machinery.

[24] Anurag Dwarakanath, Manish Ahuja, Samarth Sikand, Raghotham M. Rao, R. P. Jagadeesh Chandra Bose, Neville Dubash, and Sanjay Podder. Identifying implementation bugs in machine learning based image classifiers using metamorphic testing. *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2018.

[25] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, page 226–231. AAAI Press, 1996.

[26] Pasi Fränti and Sami Sieranoja. K-means properties on six clustering benchmark datasets. *Applied Intelligence*, 48(12):4743–4759, Dec 2018.

[27] Ana Fred. Finding consistent clusters in data partitions. In *In Proc. 3d Int. Workshop on Multiple Classifier*, pages 309–318. Springer, 2001.

[28] Brendan J. Frey and Delbert Dueck. Clustering by passing messages between data points. *Science*, 315(5814):972–976, 2007.

[29] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai2: Safety and robustness certification of neural networks with abstract interpretation. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, May 2018.

[30] Michael Hahsler. R's dbscan package v. 1.1.3, April 2019. https://cran.r-project.org/web/packages/dbscan/index.html.

[31] Greg Hamerly. *Making k-means even faster*, pages 130–140.

[32] John A. Hartigan and MA Wong. Algorithm AS 136: A K-means clustering algorithm. *Applied Statistics*, pages 100–108, 1979.

[33] Mark Hornick. Oracle r technologies overview. https://www.oracle.com/assets/media/oraclertechnologies-2188877.pdf.

[34] Lawrence Hubert and Phipps Arabie. Comparing partitions. 2:193–218, 02 1985.

[35] Janakiram MSV. Why do developers find it hard to learn machine learning?, 2017. https://www.forbes.com/sites/janakirammsv/2018/01/01/why-do-developers-find-it-hard-to-learn-machine-learning/.

[36] Jianbo Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.

[37] Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. The (black) art of runtime evaluation: Are we comparing algorithms or implementations? *Knowl. Inf. Syst.*, 52(2):341–378, August 2017.

[38] Tilman Lange, Volker Roth, Mikio L. Braun, and Joachim M. Buhmann. Stability-based validation of clustering solutions. *Neural Computation*, 16(6):1299–1323, 2004.

[39] Stuart. Lloyd. Least squares quantization in pcm. *IEEE Trans. Inf. Theor.*, 28(2):129–137, September 2006.

[40] Glenn W. Milligan and Martha C. Cooper. A study of the comparability of external criteria for hierarchical cluster analysis. *Multivariate Behavioral Research*, 21(4):441–458, 1986.

[41] Frank Nielsen. *Introduction to HPC with MPI for Data Science.* 09 2016.

[42] Nvidia. World's first functionally safe ai self-driving platform. https://www.nvidia.com/en-us/self-driving-cars/drive-platform/.

[43] Ingram Olkin. *Contributions to Probability and Statistics: Essays in Honor of Harold Hotelling.* Stanford studies in mathematics and statistics. Stanford University Press, 1960.

[44] Randal S. Olson, William La Cava, Patryk Orzechowski, Ryan J. Urbanowicz, and Jason H. Moore. Pmlb: a large benchmark suite for machine learning evaluation and comparison. *BioData Mining*, 10(1):36, Dec 2017.

[45] PAT RESEARCH. Top 15 artificial intelligence platforms in 2018, 2018. https://www.predictiveanalyticstoday.com/artificial-intelligence-platforms/.

[46] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 410–420, Prague, Czech Republic, June 2007. Association for Computational Linguistics.

[47] SAS Institute Inc. Sas/stat 12.1 user's guide, 2012.

[48] Siwakorn Srisakaokul, Zhengkai Wu, Angello Astorga, O. Alebiosu, and T. Xie. Multiple-implementation testing of supervised learning software. In *AAAI Workshops*, 2018.

[49] Douglas Steinley. Properties of the hubert-arable adjusted rand index. *Psychological methods*, 9(3):386, 2004.

[50] Gregor Stiglic and Peter Kokol. Stability of ranked gene lists in large microarray analysis studies. *Journal of biomedicine & biotechnology*, 2010:616358, 06 2010.

[51] Alexander Strehl and Joydeep Ghosh. Cluster ensembles — a knowledge reuse framework for combining multiple partitions. *J. Mach. Learn. Res.*, 3(null):583–617, March 2003.

[52] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *JMLR*, 11(Oct):2837–2854, 2010.