2016 23rd International Conference on Pattern Recognition (ICPR)
Cancún Center, Cancún, México, December 4-8, 2016

Axioms to Characterize Efficient Incremental Clustering

Sambaran Bandyopadhyay
IBM Research
sambband@in.ibm.com

M Narasimha Murty
Indian Institute of Science
mnm@csa.iisc.ernet.in

Abstract

Although clustering is one of the central tasks in machine learning for the last few decades, analysis of clustering irrespective of any particular algorithm was not undertaken for a long time. In the recent literature, axiomatic frameworks have been proposed for clustering and its quality. But none of the proposed frameworks has concentrated on the computational aspects of clustering, which is essential in current big data analytics. In this paper, we propose an axiomatic framework for clustering which considers both the quality and the computational complexity of clustering algorithms. The axioms proposed by us necessarily associate the problem of clustering with the important concept of incremental learning and divide and conquer learning. We also propose an order independent incremental clustering algorithm which satisfies all of these axioms in some constrained manner.

1 Introduction

The problem of clustering [4, 3] has been one of the central topics of machine learning. But there was not much literature which explains clustering irrespective of any particular algorithm. Most of the well-known definitions of clustering are either not mathematically rigorous or they fail to distinguish data clustering from data partitioning. Recently, some approaches have been made to propose a theoretical framework for clustering [6, 8]They provide axioms and claim that these axioms should naturally be satisfied by any good clustering algorithm. But all of these frameworks only consider the quality of the resulting clustering and no one has axiomatized the computational aspects of it.

In one step ahead, it is also very hard to define the term "efficiency" in the context of clustering. Efficiency may depend on the application or may be on the type of clustering algorithm. If we consider clustering as a computational problem, then efficiency depends on the quality and the computational aspects of clustering. In this paper we try to build an axiomatic framework for clustering which considers both the quality and the computational aspects through different axioms and hence it would try to characterize efficient clustering.

In the current era of big data analysis, the time and space complexities associated with any clustering algorithm are the key points to evaluate it. Ideally if we can design an algorithm which needs to scan each data point exactly once and requires a small memory to work with, we can easily apply it to any big data related problem. So in our framework of clustering, we capture this issue properly and address the concept of incremental learning [7]. To the best of our knowledge, a theoretical analysis of incremental clustering has not been addressed so far. Not only that, we bring another useful learning strategy known as divide and conquer to our clustering framework. It permits parallelism. Additionally it would be better if the algorithm satisfies some more desirable properties of clustering. All of these concepts will be formally explained in the subsequent sections.

Thus in this paper, we construct an axiomatic framework for incremental clustering, which contains both the quality and computational aspects of clustering. We bring the concepts of divide and conquer learning and deletion of data instances from clustering in our axiomatic framework and fit them in the incremental clustering paradigm. We propose an order independent incremental clustering algorithm to show the existence of such an algorithm which can satisfy all the axioms of our framework in some constrained manner.

2 Background and Related Work

As discussed earlier, there are some axiomatic frameworks for clustering present in the literature. According to Jon Kleinberg [6], any clustering algorithm should satisfy three axioms namely Scale Invariance, Richness and Consistency. To make the paper self-contained, we give a brief description of these axioms.

978-1-5090-4846-5/16/$31.00 ©2016 IEEE

Scale Invariance ensures that scaling the distance function by any arbitrary positive constant does not change the resulting clustering. **Richness** ensures that a clustering algorithm is able to produce all possible partitions of the data set. The last axiom **Consistency** ensures that reducing (or keeping them same) the intra cluster distances and increasing (or keeping them same) the inter cluster distances among the data points of a clustering would not change the resulting clustering.

But in the same paper, he has shown that there is no algorithm which can satisfy all these properties simultaneously. So instead of Richness, we use the property **k-richness** [8], which says that a clustering algorithm should be able to produce all possible k-partitions of the data set. It can be easily noticed that, all of these properties are dealing with the end product of clustering and none of them has mentioned anything about the way clustering algorithms should produce them and the associated computational complexity. We have addressed this research gap in this paper.

To make the clustering algorithms computationally efficient, we use the concepts of incremental learning and divide and conquer paradigm in clustering. But it is also known that incremental learning suffers from order effects [7]. We already know some incremental clustering algorithms (for example Leader, Birch [4]) and incremental frequent patterns mining algorithms based on PC-Tree [1]. But to the best of our knowledge, there is no efficient order independent incremental clustering algorithm present in the literature.

3 Efficient Clustering

As discussed earlier, efficiency depends on the quality and computational aspects of clustering. To discuss the computational aspects of clustering, we bring following two important concepts.

3.1 Incremental Clustering

According to [7], a Learner L is **Incremental** if (i) it processes one training example at a time, (ii) does not reprocess any previously seen training example and (iii) retains only one knowledge structure in the memory. Clearly incremental algorithms scan the input data set at most once. It is also easy to use incremental algorithms in online setting. A clustering algorithm is called an **Incremental Clustering Algorithm** if it satisfies the three conditions of an incremental Learner.

A **Distance Function** is defined as $d : \chi \times \chi \longrightarrow R$, where χ is the universal set of all instances (or data points) and R is the set of Real numbers, such that, 1. $d(x, y) \geq 0$, the equality holds if and only if $x = y$ and 2. $d(x, y) = d(y, x)$ Let $X = \chi^n$, where n is the number of instances. The **Clustering Function** can be defined as, $f_d : X \longrightarrow \Delta$, where Δ is the set of all possible k partitions of a set of n instances. Here f_d basically takes an ordered set of n instances and clusters them to get a k-partition. We have put the distance function d in the subscript of f for notational convenience.

Generally clustering algorithms maintain a data structure (or knowledge structure) in the memory which is updated in each iteration of the algorithm. We call this data structure as **Abstraction** in this paper. Abstraction can typically be the set of intermediate clusters formed during the iterations of the algorithm. Here we use $\mathcal{A}$ to denote the set of all abstractions.

Let us divide the whole process of clustering into two sub-process, $h_d : \mathcal{A} \times \chi \longrightarrow \mathcal{A}$ and $g_d : \mathcal{A} \longrightarrow \Delta$, and viw it as follows. First there is some initial abstraction (which can be empty) present in the memory. The function h_d iteratively modifies the abstraction after scanning a data instance. We get the final abstraction in the memory after this scanning procedure ends. The function g_d then maps this abstraction to the corresponding clustering (or produce the corresponding cluster representatives). So in this paper we can say that a clustering function is **incremental** if it can produce the final abstraction and the cluster representatives incrementally [7]. But it is also important for an incremental algorithm to be order independent. An incremental clustering algorithm is **Order Independent** if irrespective of any ordering of the data instances, the algorithm can produce the same clustering.

3.2 Divide and Conquer Clustering

Here we divide the whole set of data instances into C partitions. We apply the function h_d on each of these parts to get C different abstractions. Then we make $C/2$ pairs of abstractions from the set of C abstractions (assuming C is divisible by 2) and **merge** (as defined in Section 4) the two abstractions in a pair to get a new abstraction. In this way we get a set of $C/2$ abstractions. We continue the same procedure until we get the final abstraction. Then we apply the function g_d on the final abstraction to get the resulting clustering. If the merging of any two abstractions is independent from other merging operations, we can exploit a huge amount of parallelism in this process.

4 An Axiomatic Framework

In this section, we describe some desirable properties of an efficient clustering algorithm. Although, axiom in general may imply an intrinsic property of an object,

but in this work, we have used 'axiom' and 'property' interchangeably. Mainly Properties 1 to 4 deal with the computational issues and Properties 5 to 7 will capture the qualitative issues of clustering.

As described earlier, we maintain an abstraction in the main memory and modify this abstraction as the instances are coming. But the modification procedure should not involve any other previously seen instance. Otherwise the time to scan other instances would significantly increase the total time to insert the current instance to the present abstraction. So if the total number of patterns in the input is n, we can order the patterns as x_1, x_2, $x_3 \cdots$, x_n and the memory abstractions as A_0, A_1, $A_2 \cdots$, A_n. It is to be noted that patterns in the input need not be unique.

Axiom 1 ***Insertion Property:*** *To generate A_{k+1}, it is sufficient to know only A_k and x_{k+1}. $h_d(A_k, x_{k+1}) = A_{k+1}, \forall k = 0, 1, 2, \cdots \forall x_{k+1} \in \chi, \forall A_k, A_{k+1} \in \mathcal{A}$*

One major problem with most incremental learning algorithms is order dependency. For example, in center based clustering like incremental k-means [3], updating centers incrementally introduces dependency on the order in which the input instances are processed. If our algorithm is order dependent, its performance may vary significantly on different ordering of the instances and also it would be hard to analyze its performance irrespective of any order. We present the basic axiom for order independence in terms of two instances and will generalize it in Claim 2.

Axiom 2 ***Swapping Property:*** *$\forall x, y \in \chi$ and $\forall A \in \mathcal{A}$, $h_d(h_d(A, x), y) = h_d(h_d(A, y), x)$*

We want to incorporate the operation of deletion of a previously inserted instance from the present memory abstraction. Deletion itself can be a very common requirement in clustering. It is specially relevant to an incremental set up where instances are coming one by one. So we want to have the following definition which characterize deletion operation formally.

Definition 1 ***Deletion:*** *Deletion is a function (or operation) which takes one abstraction and a valid instance and outputs an abstraction such that the effect of that instance is removed from the first abstraction. We will call it as the deletion of that instance from the abstraction. An instance will be* ***valid*** *if it has been inserted into the abstraction earlier but not deleted yet. Let us denote the deletion of some valid instance x_j from the current abstraction A_k as: $h'_d(A_k, x_j)$*

$$= h'_d(h^k_d(\ A_0, x_1; x_2; \cdots; x_{j-1}; x_j; x_{j+1}; \cdots x_k\ ), x_j)$$
$$= h^{k-1}_d(A_0, x_1; x_2; \cdots; x_{j-1}; x_{j+1}; \cdots x_k)$$

where A_0 is the initial abstraction, A_k is the abstraction formed after inserting the k instances $x_1; x_2; \cdots; x_{j-1}; x_j; x_{j+1}; \cdots x_k$ in order and h^k_d means applying the function h_d k-times consecutively.

One trivial way to delete any instance from the abstraction (or clustering itself) is to exclude that instance from the input set of instances and then run the whole clustering algorithm on the rest. But this process would be extremely inefficient because it scans all the instances every time we want to delete an instance. So we want the deletion operation in our framework to be efficient in the sense that the process only requires the present memory abstraction and the instance to be deleted. It should not need any previously seen instance.

Axiom 3 ***Invertible Property:*** *For a given $x \in \chi$, we can uniquely determine A_1 from A_2 where $h_d(A_1, x) = A_2$, $\forall A_1, A_2 \in \mathcal{A}$, without scanning any other instance.*

This is called Invertible Property as we can efficiently find the inverse image of A_2 under some fixed instance x. This property would ensure efficient deletion as shown in Lemma 3.

Now we will try to axiomatically capture the notion of Divide and Conquer strategy discussed in Section 3.2.

Definition 2 ***Merge:*** *Merge is a function (denoted by $\circ$) which takes two abstractions A^1 and A^2, and produce another abstraction, $A^1 \circ A^2 = A^3$, with the following condition.*
Suppose $\mathbf{x} \in X$ and $\mathbf{x} = x_1 x_2 \cdots x_k x_{k+1} \cdots x_n$. Let, $\mathbf{x^1} = x_1 x_2 \cdots x_k$ and $\mathbf{x^2} = x_{k+1} \cdots x_n$. Say, A_0 is the initial abstraction. $h^k_d(A_0, \mathbf{x^1}) = A^1$ and $h^{n-k}_d(A_0, \mathbf{x^2}) = A^2$, then $h^n_d(A_0, \mathbf{x}) = A^3$

So merge operation says that we can divide the data set into two blocks and generate two abstractions by running the clustering functions separately on them. Then we can merge these two abstractions such a way that the resulting final abstraction is equivalent to the one generated by running the algorithm on the whole data set. But again the merging operation can be done trivially by running the clustering algorithm on the whole data set. But we want to make the merging operation efficient by the following axiom.

Axiom 4 ***Conquer Property:*** *To merge any two abstractions, it is sufficient to use only those two abstractions. In other words, we do not need to scan any other instance to merge those two abstractions.*

Now we want to present three axioms which can be used to judge the quality of the resulting clustering. They are well-explained in the literature and are also

Algorithm	Insertion	Swapping	Invertible	Conquer	Scale Invariance	k-rich	Consistency
Leader	Yes	No	No	No	No	Yes	No
k-means	No	Yes [1]	No	No	Yes	Yes	No
Incremental k-means	Yes	No	No	No	Yes	Yes	No
Single Linkage	No	Yes	No	No	Yes	Yes	Yes
Tree Clustering Algorithm	Yes	Yes	Yes	Yes	No [2]	Yes	No [2]

Table 1. Different Clustering Algorithms Satisfying Some of the Axioms in our Framework

briefly discussed in Section 2. Here we will rephrase them accordingly to fit into our framework.

Axiom 5 ***Scale Invariance:*** $h_d^n(A, x^n) = h_{d'}^n(A, x^n)$, *whenever* $d' = \alpha.d$, *where* α *is any positive real number and* x^n *as the sequence of n instances.*

Axiom 6 ***Consistency:*** $h_d^n(A, x^n) = h_{d'}^n(A, x^n)$, *where d' is a* $\Gamma - transformation$ *of d [6].*

Axiom 7 ***k-richness:*** $\{g_d(A) : A \in \mathcal{A}\} = \Delta$ = *Set containing all k-partitions of the n instances.*

Table 1 shows some well-known clustering algorithms [4] which satisfy some of the axioms in our framework. It is possible to characterize most of the clustering algorithms with the help of these axioms. Following lemmas are given to formally relate the axioms with the related concepts [3].

Lemma 1 h_d ***is incremental if and only if Axiom 1 is satisfied.***

Lemma 2 ***Suppose*** h_d ***is incremental. Then it would be Order Independent if and only if Axiom 2 is satisfied.***

The "if" part above lemma can be proved by using principle of mathematical induction. The "only if" part is direct as Property 2 is the same as order independence when the number of instances is 2.

Lemma 3 ***If*** h_d ***is incremental and Order Independent then deletion can also be performed efficiently (incrementally) if and only if Axiom 3 is satisfied.***

We call a Divide and Conquer Clustering (as discussed in Section 3.2) **Efficient** if the merging operation can be done just based on the two given abstractions and without scanning any other instance. Following lemma connects it with our framework.

[1] k-means satisfies Swapping property under the assumption that the initialization of centroids are fixed for a data set

[2] As discussed in Section 5, if we restrict the distance functions only to be SPWDF, Tree Clustering will also satisfy Scale Invariance and Consistency properties

[3] **Due to page limitation, all the proofs are given in the supplementary material**

Lemma 4 ***A clustering algorithm is Efficient Divide and Conquer algorithm if and only if it satisfies Axiom 4.***

Hence from the above lemmas along with the Axioms 5, 6, 7, we can reach to the following theorem.

Theorem 1 ***Axioms 1 to 7 are necessary and sufficient conditions for a clustering algorithm to be order independent incremental and Efficient Divide and Conquer algorithm having efficient deletion operation, which also satisfies Scale Invariance, Consistency and k-richness.***

5 Existential Result: Tree Clustering

Now we would show that it is possible to design some algorithm which satisfies all the axioms proposed in our framework, may be in some constrained manner. For this purpose, we are going to construct a Clustering Algorithm based on FP Tree [5]. We call this algorithm Tree Clustering Algorithm.

5.1 Preliminary Setup for the algorithm

We represent any data instance $x \in \chi$, as an ordered set (string) of some items, such as, $x = acdef$. For each string, we mark the positions as $0, 1, 2 \cdots$, where item at $0th$ position corresponds to the first feature of the data instance and so on. Let these items $a, b, c, d, e, f, \cdots \in I$, where I is an ordered finite set called Item Set. So for any two given items, we can always find out which one comes first in I. We can always assume that blank symbol (or blank item) ϵ belongs to the $0th$ position in I. Let us also assume that ϵ refers to the absence of a feature value and it can occur (may be in multiple numbers) only after all other non-blank items of the instance. We consider maximum possible length of any instance is a constant.

Now when we want to compare two instances x_1 and x_2, there may be a match at position i if $x_1(i) = x_2(i)$, else there is a mismatch which can be of two types.

Definition 3 *If* $x_1(i) \neq \epsilon$ *and* $x_2(i) \neq \epsilon$ *and also* $x_1(i) \neq x_2(i)$, *then it is called a* ***strong mismatch*** *at*

position i. If either one of $x_1(i)$ or $x_2(i)$ is equal to ϵ and the other one is not, then it is called a ***weak mismatch*** *at position i.*

Now we introduce a particular class of distance functions which is needed to prove the existential result.

Definition 4 *A distance function is called a Strongly Prefix Weighted Distance Function (**SPWDF**) if all of the following 3 conditions are satisfied, $\forall x, y, x_1, x_2, y_1, y_2 \in \chi$. Here len(.) returns the number of non-blank items of an instance.*
*1. **Additive Distance:** $d(x, y) = d(x(0, i), y(0, i)) + d(x(i + 1, l), y(i + 1, l))$; where $l = max(len(x), len(y))$, The sub instance $x(i, j)$ denotes the instance formed by taking the items from position i to j of the instance x.*
*2. **Item Set Consistent:** $d(x(i, i), y(i, i))$ increases if the distance between the ith items in x and y is increased in the item set I and vice versa.*
*3. **Strong Prefix Weightage:** If $d(x_1(0, i), x_2(0, i)) = d(y_1(0, i), y_2(0, i))$ and*
I. If match at (i+1)th position between x_1 and x_2 and strong mismatch at (i+1)th position between y_1 and y_2, then $d(x_1, x_2) < d(y_1, y_2)$.
II. If weak mismatch at (i+1)th position between x_1 and x_2 and strong mismatch between (i+1)th or some higher position of y_1 and y_2, then $d(x_1, x_2) < d(y_1, y_2)$.
III. If match at (i+1)th position between x_1 and x_2, weak mismatch between (i+1)th position of y_1 and y_2 and there is no Strong mismatch onwards in x_1 and x_2, then $d(x_1, x_2) < d(y_1, y_2)$.

Clearly SPWDF gives more importance to the prefix part of an ordered instance and more weight to a strong mismatch than a weak mismatch. An important property of SPWDF is that, if a distance function d belongs the class of SPWDF, then it would be possible to rank n data instances based on their distance d from a given data instance without knowing the exact definition of d (just by checking strong and weak mismatches).

Example 1 *Let us consider four instances $x_1 = abcde\cdots$, $x_2 = ab'cde$, $x_3 = abc'd'e'\cdots$ and $x_4 = a$. Here b' means some non-blank symbol other than b. Similar is for c', d' and e'. If the distance function satisfies SPWDF, then $d(x_1, x_2) > d(x_1, x_3) > d(x_1, x_4)$.*

5.2 Description of Tree Clustering Algorithm

The first part of our algorithm is to build a Tree Structure which we call **Incremental Frequent Pattern Tree (IFPT)**. The generation process of IFPT is similar to that of FP Tree. But we do not need to sort the items of an instance (as defined in the previous subsections) and we do not need to have pointers between same items at different paths of the Tree. This help us to generate the Tree incrementally using only a single scan over the data set. Similar to FP Tree, each node in this tree contains two fields. First one is for carrying the item name and the second field is to count the occurrence of that item in that path.

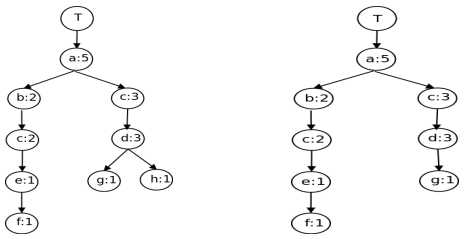

Figure 1. Tree abstractions

In this algorithm, the input is an ordered set of n instances, each of which is as defined earlier. At first, there is just a single starting node (root) T as the initial abstraction. When the first instance comes, it creates a node for each non-blank item of it in the order and places as a branch in the tree. The node containing the first item becomes a child of the root node of the tree. When the next instance comes, it tries to find the branch with maximal prefix match in the existing abstraction. Say it finds a branch where there is the maximal match up to item i. Then it will increase the counter by 1 for the first i nodes of that branch and then create a new child at the ith node and put the remaining item nodes consequently with counter value 1. In this way we will build an IFP Tree from the set of instances. We call this tree as tree abstraction. Then we will try to find k cluster representatives, $k \leq n$ as there in Algorithm 1.

To get the final clustering, we assign each instance to its nearest representative measured by SPWDF. It is equivalent as comparing the prefix part of the instance to the cluster representatives and assigning it to where the prefix matching is maximum in length. We give an illustrative example here.

Example 2 *Consider the following instances: $x_1 = abcef$; $x_2 = abc$; $x_3 = acd$; $x_4 = acdg$; $x_5 = acdh$ First we build the IFPT and get the abstraction shown in the left part of Figure 1. Here $m = 3$ for this IFPT. Now suppose we set $k = 3$. So we consider each path as a representative of a cluster. So we assign x_1 and x_2 in a cluster corresponding to the representative $abcef$, x_3 and x_4 in the cluster corresponding to the representative $acdg$ and x_5 in the cluster corresponding to the representative $acdh$. If the value of k is 2. So we need 2*

representatives. We modify the tree according to the algorithm to get the tree as shown in the right one of Figure 1. The assignment of instances to the clusters will again be based on maximal prefix matching or equivalently by nearest distance measured by SPWDF.

Following theorem shows that there exist some clustering algorithm which satisfies all the axioms (satisfiability of the Axioms 3 and 4 has been addressed in Section 5.3) of our framework in some constrained manner.

Algorithm 1 Tree Clustering Algorithm

Input: Set of data instances, number of clusters k
Output: k cluster representatives
Build the IFP Tree from the set of instances.
Say M = Set of paths from root to leaf nodes, $|M| = m$
while $m > k$ **do**
    From the set M, take the two paths of minimum distance (based on SPWDF) from each other and delete the path from M which lexicographically comes later;
    $m \leftarrow m - 1$
end while
while $m < k$ **do**
    Choose the node N such that N belongs to at least one path from the set M, and the count value of N is greater than the sum of the count values of its children
    If there are more than one of such nodes, choose the node N with the least level (level of root is 0) among them
    Insert the path from root T to N into M, and $m \leftarrow m + 1$
end while
Return all the selected cluster representatives

Theorem 2 ***If we allow feasible distance functions only from SPWDF, then Algorithm 1 is an order independent incremental clustering algorithm satisfying Scale Invariance, k-richness and consistency.***

To the best of our knowledge, this is the first order independent incremental clustering algorithm which also satisfies other conditions of efficiency. Even for a general class of distance, Tree clustering would satisfy all the axioms except Scale Invariance and Consistency.

5.3 Efficient Deletion and Divide-Conquer

Efficient Deletion: We have presented the Algorithm 2 to delete any valid instance from an abstraction. Please note that while deleting any valid instance from the abstraction, we never scan any other instance. **So deletion in this case satisfies Axiom 3 in section 4.**

Efficient Divide and Conquer: The basic concept is to divide the whole data set into several parts and use Tree Clustering algorithm for each part to generate separate tree abstractions independently. Then we can merge different tree abstractions using Algorithm 3 which does not scan any other instance while merging two abstractions. **So it satisfies Axiom 4 in Section 4**.

Algorithm 2 Deletion Algorithm

Input: Tree abstraction T and an instance x to delete
Output: Modified abstraction T after deleting x (if it is valid)
Starting from the children of the root node T, find a path which matches with the given instance x
If we get the match for the whole instance, reduce the count of each node of the matching path by 1.
If we do not get a match, output that the instance x is not valid.

Algorithm 3 Merge Abstractions

Input: Two tree abstractions indexed by root pointers $T1$ and $T2$
Output: Merged abstraction $T1$
for Each child node of $T2$ **do**
    If the item i of the child node is NOT in a child of $T1$, create a new child at $T1$ which will point to the whole subtree rooted at i of $T2$.
    Else increase the count of i at $T1$ by the count of i at $T2$ and call recursively Merge Abstractions function with input pointers at the node i of $T1$ and the node i of $T2$.
end for

6 Conclusions and Future Work

In this paper, we have proposed an efficient clustering framework which is very important in characterizing different clustering algorithms. We also show the existence and propose a clustering algorithm which satisfies all the axioms of our framework in a constrained manner. A through experimental evaluation of the algorithm can be done in some future work.

References

[1] V. S. Ananthanarayana, M. N. Murty, and D. K. Subramanian. An incremental data mining algorithm for compact realization of prototypes. *Pattern Recognition*, 34(11):2249–2251, 2001.

[2] S. Bandyopadhyay, R. Narayanam, P. Kumar, S. Ramchurn, V. Arya, and I. Petra. An axiomatic framework for ex-ante dynamic pricing mechanisms in smart grid, 2016.

[3] W. Barbakh and C. Fyfe. Online clustering algorithms. *International Journal of Neural Systems*, 18(03):185–194, 2008.

[4] R. Duda, P. Hart, and D. Stork. Pattern classification. 2nd. *Edition. New York*, 2001.

[5] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data mining and knowledge discovery*, 8(1):53–87, 2004.

[6] J. M. Kleinberg. An impossibility theorem for clustering. In *NIPS*, pages 446–453, 2002.

[7] P. Langley. Order effects in incremental learning. *Learning in humans and machines: Towards an interdisciplinary learning science. Pergamon*, 136:137, 1995.

[8] R. Zadeh and S. Ben-David. A uniqueness theorem for clustering. In *UAI*, pages 639–646, 2009.