Streaming Approach to Schema Profiling

Streaming Approach to Schema Profiling

Chiara Forresi[0000−0001−5652−2455], Matteo Francia[0000−0002−0805−1051], Enrico Gallinucci[0000−0002−0931−4255] ✉, and Matteo Golfarelli[0000−0002−0437−0725]

University of Bologna, Cesena, Italy
{chiara.forresi,m.francia,enrico.gallinucci,matteo.golfarelli}@unibo.it
✉ Corresponding author

Abstract. Schema profiling consists in producing key insights about the schema of data in a high-variety context. In this paper, we present a streaming approach to schema profiling, where heterogeneous data is continuously ingested from multiple sources, as is typical in many IoT applications (e.g., with multiple devices or applications dynamically logging messages). The produced profile is a clustering of the schemas extracted from the data and it is computed and evolved in real-time under the overlapping sliding window paradigm. The approach is based on two-phase k-means clustering, which entails pre-aggregating the data into a coreset and incrementally updating the previous clustering results without recomputing it in every iteration. Differently from previous proposals, the approach works in a domain where dimensionality is variable and unknown apriori, it automatically selects the optimal number of clusters, and detects cluster evolution by minimizing the need to recompute the profile. The experimental evaluation demonstrated the effectiveness and efficiency of the approach against the naïve baseline and the state-of-the-art algorithms on stream clustering.

Keywords: Schema profiling · Stream clustering · K-means · Approximation algorithm.

1 Introduction

Recent years have shown an increasing interest in the streaming paradigm, which gives users the possibility to monitor and analyze in near real-time a never-ending flow of data. This interest has been pushed by Internet-of-Things and Big Data applications, where a multitude of sources can generate huge quantities of data in a continuous manner. Depending on the application, the variety of data flowing in a stream can be significant. Real-world examples include applications collecting logs without fixed structures from multiple sources [7] or IoT systems collecting messages from a variety of devices, each with its own features and measurements [6]. In such scenarios, technicians are interested in characterizing the incoming traffic and knowing which kinds of error messages or measurements are currently flowing in the stream; this can be done by analyzing the high level of heterogeneity in the schemas of the messages, which conveys information about the nature of the data. The family of techniques that produce key insights about

the intensional aspect of data (i.e., the schema) is known as *schema profiling* and has been applied in the past to discern the variety within schemaless collections in NoSQL databases [7], but it has never been applied in streaming applications.

In this paper, we propose a streaming approach to schema profiling, with the goal to produce real-time insights about the schemas of data elements in a streaming application. The approach is based on the overlapping sliding window paradigm and the produced *profile* consists in clustering the schemas (extracted from the elements) in the window, which will help the user paint a high-level picture of the schema variety in the stream. The reference clustering algorithm is k-means, which must be adapted to the streaming application. Our proposal is a two-phase algorithm [16] to be run on every window. The first phase consists in building a *coreset* of pre-aggregated schemas by relying on a compact data structure (called *summaries*). In the second phase, the profile is built by clustering the summaries in the coreset or by updating the profile obtained in the previous iteration; here, we introduce rules to incrementally detect changes in the (number of) clusters and consider a reclustering policy to recompute clustering if the rules do not improve the result.

Several approaches have been proposed so far to carry out k-means clustering on a stream of data [16], which essentially introduce some degrees of approximation to compensate for the need of computing clustering in real-time; most importantly, approximation is introduced by pre-aggregating the original data elements and/or by incrementally updating the clustering result, without recomputing every time that the window slides. However, existing works are not applicable to our context, mostly because they either have limited capabilities to capture the evolution of clusters in time or they are designed for a Euclidean space with fixed dimensionality (as Euclidean distance loses significance in a variable high-dimensional space). Most algorithms work under the same two-phase method used in this paper. The most influential work about pre-aggregating data into a compact data structure is BIRCH [15], whose clustering features have been widely adopted and extended by related works. CSCS [14] is the one most similar to our. Differently from our approach, CSCS (i) requires a fixed predefined number of clusters k, (ii) uses a compaction technique that works only with data in a fixed domain (whereas the domain of attributes is not known apriori), and (iii) does not capture all kinds of cluster evolution. The most recent work on streaming k-means clustering supporting a variable number of clusters is FEAC-S [2]; however, FEAC-S does not involve a pre-aggregation of the data (it directly computes and updates clusters, which is more time-consuming) and, even though it formally supports all kinds of cluster evolution, it relies on pseudo-random criteria to decide when the evolution operations should be applied. Overall, no streaming k-means approaches support a variable and unknown dimensionality. A related field is the one of continuous profiling which generically consists in continuously monitoring the evolution of properties of some data properties [12]; however, existing approaches in this area focus on other aspects like functional dependencies [4] and data stream classification [13]. Real-time profiling has been proposed to dynamically extract patterns

from unstructured logs [5], which have no notion of schemas. To the best of our knowledge, this is the first work to address the problem of profiling schemas in a streaming context.

The outline of the paper is as follows. Section 2 introduces the basic concepts used in the paper. Section 3 presents our approach, which is experimentally evaluated in Section 4. Conclusions are drawn in Section 5.

2 Basic concepts

In this work, the first citizen is the schema of a message in the stream. Streaming applications typically exchange messages in the form of JSON documents, i.e., collections of key-value pairs, where keys are the schema elements (i.e., *attributes*). It is common to refer to nested attributes with *paths*, i.e., a dot-concatenation of the nested key with all the parent keys. The schema information that we are interested in is the union of all paths appearing in a document.

Definition 1 (Schema). *A schema is a set of attributes* $A = \{a_1, a_2, \ldots, a_n\}$*, where* a_i *is a string representing the attribute's path.*

The real-time analysis of the documents' schemas being ingested in a stream is carried out under the sliding window paradigm: the analytical algorithm is designed to run on a subset of stream elements (i.e., the window), which progressively moves in time to discard older elements and include newer ones. The *window* is typically defined by two parameters [1]: the *window length* (i.e., the amount of data to include in the window) and the *window period* (i.e., the frequency of the algorithm's execution). Both of these parameters can be expressed in terms of either time or the number of elements. For the sake of simplicity, we will use the number of elements. In our setting, we consider overlapping windowing with the window length (i.e., w_len) being set as a multiple of the window period (i.e., w_per), so that both the stream and the window can be defined in terms of *panes*, i.e., smaller subsets of stream elements of length w_per.

The clustering algorithm we take into account is k-means++ [3] (i.e., a more efficient version than traditional k-means with a non-random initial setting, which leads to a faster convergence). It is a type of unsupervised learning providing a straightforward method to categorize a given dataset into k clusters, where k is a user-defined parameter and each cluster is represented by a *centroid* (usually computed as the average of all the elements in a cluster).

Definition 2 (Clustering result). *Given a dataset of elements* U*, a clustering result is a set of clusters* $C = (c_1, c_2, \ldots, c_k)$ *where* $c_i = \langle E, \hat{e} \rangle$*;* $c_i.E \subset U$ *is the set of elements assigned to* c_i *and* $c_i.\hat{e}$ *is the centroid.*

The objective function of k-means is to find a solution C with minimum *cost*, meant as the sum of squared distances (according to a distance function d, usually Euclidean) between each element $e \in c_i.E$ and the centroid of its assigned cluster (i.e., $c_i.\hat{e}$). The algorithm starts by selecting k centroids from

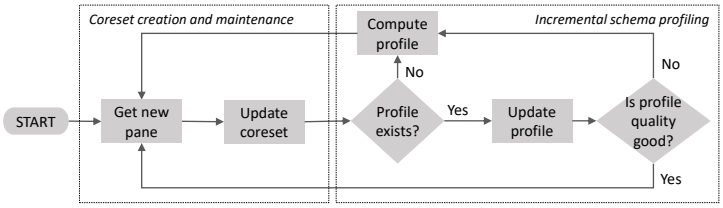


Fig. 1: Overview of the real-time schema profiling approach.

U and assigns each element to the nearest centroid; then, the centroids are updated based on the assigned elements, and the procedure is iteratively repeated until convergence (which occurs when elements are not reassignments or when a maximum number of iterations is reached).

In this work, k is not known apriori; thus, we rely on the Ordered Multiple Runs of k-Means approach [11] (here called OMRk++), which consists in running k-means++ for all values of k between 2 and $\sqrt{|U|}$ and choosing the solution with the value of k that maximizes the *simplified silhouette* (SS). The SS of an element e assigned to cluster c_i (i.e., $e \in c_i.E$) is calculated as $SS(e) = \frac{y(e)-x(e)}{max(x(e),y(e))}$, where $x(e) = d(e, c_i)$ is the distance between the element and its centroid, and $y(e) = \arg\min_{c \in C \setminus c_i} d(e, c)$ is the minimum of the distances between the element and the other centroids. The SS of the clustering solution is the average of the values for every element, i.e., $SS(C) = avg_{e \in U} SS(e)$.

3 Real-time schema profiling

Schema profiling [7] aims to provide insights into the structure of the schemas in a given dataset. In the context of this paper, the profile we aim to provide consists in grouping schemas into a non-predefined number of clusters with the OMRk++ technique (i.e., the clustering result in Definition 2), which allows to easily distinguish the main kinds of schema in the data. In real-time schema profiling, the challenge is to provide this result on a stream of continuously incoming data, shifting the focus on the latest data window and keeping the stream's pace. The naïve way to produce the intended result is to simply run OMRk++ on each data window. This is obviously inefficient and inapplicable in a streaming context, as (i) the clustering solution would be recomputed from scratch each time, without considering the previous result and analyzing each stream element multiple times (depending on the window settings), and (ii) the clustering algorithm would run on a (potentially very) large quantity of elements.

The approach proposed in this paper is called DSC (Dynamic Stream Clustering) and consists in returning an approximate solution to the problem, which uses a fixed amount of memory and runs with very high performances. The approximations enabling a faster execution are the following.

1. The cardinality of the data in input to the clustering algorithm is reduced to a *coreset*, i.e., a meaningful set of summaries (defined in Section 3.1). By clustering fewer data, DSC is able to converge in shorter times. The coreset is maintained in time by removing expired summaries and adding new ones.
2. Instead of clustering the data in the coreset for every window , DSC incrementally updates the clustering result by reflecting the changes observed in the coreset. Unlike previous work, DSC captures multiple evolution phenomena of clusters including changing the number of clusters.

The approach is structured in two phases, sketched in Figure 1. In the coreset creation and maintenance phase, a new pane of data is retrieved from the stream and the coreset is updated (i.e., summaries that are not anymore in the window are removed, and new summaries are added). In the incremental schema profiling phase, the profile is created by running OMRk++ on the coreset (if no profile existed yet) or updated. Remarkably, the update of the profile goes beyond the simple update of centroids after the removal of expired summaries and the assignment of new ones, but it includes the detection of changes in the number of clusters (i.e., if two clusters need to be merged, or if a cluster needs to be split). To prevent the updated profile from drifting too much from the optimal solution (i.e., the execution of OMRk++ on the coreset), we adopt heuristics to decide if the profile has to be recreated again from scratch.

3.1 Clustering schemas with summaries

Two-phase stream clustering algorithms rely on a compact data structure that summarizes a set of stream elements. The most notable is the *clustering feature*, introduced in BIRCH [15] and used/extended in several related works [16]. In this paper, we introduce the notion of *summary*.

Definition 3 (Summary). *A* summary *of a multiset of schemas $\mathcal{A}$ is a compact structure $s^{\mathcal{A}}$ giving a fuzzy summarization of the attributes found in $\mathcal{A}$. Given an attribute a, $s^{\mathcal{A}}.p(a) \in [0,1]$ is the percentage of schemas in $\mathcal{A}$ that contain a. The number of schemas summarized by $s^{\mathcal{A}}$ is $s^{\mathcal{A}}.card = |\mathcal{A}|$. The union of the attributes in $\mathcal{A}$ is a set indicated with $s^{\mathcal{A}}.A$.*

Like clustering features, summaries are additive, i.e., $s^{\mathcal{A}_i+\mathcal{A}_j} = s^{\mathcal{A}_i} + s^{\mathcal{A}_j}$. The proof is omitted for space reasons. To simplify the notation, the superscript referring to the multiset of schemas summarized by s is omitted when not needed.

The additivity of summaries allows defining the centroid of a cluster of summaries as the sum of all the summaries in the cluster.

Definition 4 (Centroid). *Let $c_i = \langle S, \hat{s} \rangle$ a cluster of a multiset of summaries $c_i.S$; then, the centroid is $c_i.\hat{s} = \sum_{s \in S} s$.*

In most k-means applications, the distance function typically used is the Euclidean one. In our context, clustering must be carried out on summaries of elements (i.e., schemas) whose dimensionality (i.e., the set of represented attributes) is not known apriori. As we recall, attributes are simply strings with

no formal limitation and no order, i.e., it is not a Euclidean space. Thus, to measure the distance between two summaries we rely on the Jaccard distance [10], which enables the comparison of two elements independently of their dimensionality. Given two generic sets A and B, the Jaccard similarity is defined as $sim_J(A,B) = \frac{|A \cap B|}{|A \cup B|}$, and the distance is derived as $d_J(A,B) = 1 - sim_J(A,B)$. Since summaries are fuzzy, we adopt its weighted variant [8].

Definition 5 (Weighted Jaccard similarity and distance). *Given two summaries s_i and s_j, their weighted Jaccard similarity is: $sim_{WJ}(s_i, s_j) = \frac{\sum_{a \in s_i.A \cup s_j.A} min(s_i.p(a), s_j.p(a))}{\sum_{a \in s_i.A \cup s_j.A} max(s_i.p(a), s_j.p(a))}$. The weighted Jaccard distance is: $d_{WJ}(s_i, s_j) = 1 - sim_{WJ}(s_i, s_j)$.*

The purpose of using summaries is to group together similar schemas (based on sim_{WJ}) in order to build the coreset given in input to the clustering algorithm. Each summary of similar schemas is associated with an *id* and stored in a *bucket*. To support deletion/insertion of old/new summaries, buckets contain the list of summaries in different panes with the same *id*; then, the clustering algorithm run on the *representative* summary of each bucket.

Definition 6 (Coreset, bucket). *A* coreset *B is a set of at most $2m$ buckets. A* bucket *$b \in B$ is defined as $b = \langle id, S, r \rangle$, where $b.id$ is an identifier, $b.S$ is the list of summaries (one for each pane of the window) whose schemas are identified by $b.id$, and $b.r$ is the representative summary of b, corresponding to the sum of the summaries in $b.S$ (i.e., $b.r = \sum_{s \in b.S} s$). The threshold m regulates the size of the coreset, reducing the number of buckets to m when it exceeds $2m$.*

3.2 Incremental schema profiling

In the incremental schema profiling phase of the algorithm, the coreset is used to produce the clustering profile of the current window of schemas. In the very first iteration, this is done by running OMRk++ on the summaries in the coreset, which initially consists of one pane alone. In the next iterations, the current profile is incrementally updated in accordance with the updates in the coreset. A reclustering policy is adopted to prevent the updated profile from drifting too much from the optimal solution (i.e., the execution OMRk++ on the coreset); when activated, the current profile is recreated from scratch by running again OMRk++ on the summaries in the coreset. When (re)creating the profile, the input data U to OMRk++ consists of the set of representatives of each bucket, i.e., $U = \{b.r : b \in B\}$; then, each bucket b is associated with a single cluster.

The challenge lies in updating a previous profile incrementally, aiming to approximate the profile that could be generated from scratch using the current window's data as closely as possible. This task is complex and subject to the concept-drift problem, which is usually dealt with by related works through a reclustering policy: it consists in evaluating the quality of the current solution and deciding whether it is still acceptable or must be recomputed. For instance, [14] measures the Kullback-Leibler divergence [9], which measures how the new

solution represents the old data; when the loss in precision goes beyond a given threshold, the reclustering policy is activated. In the following, we enumerate the phenomena impacting a clustering solution in an evolutionary streaming context.

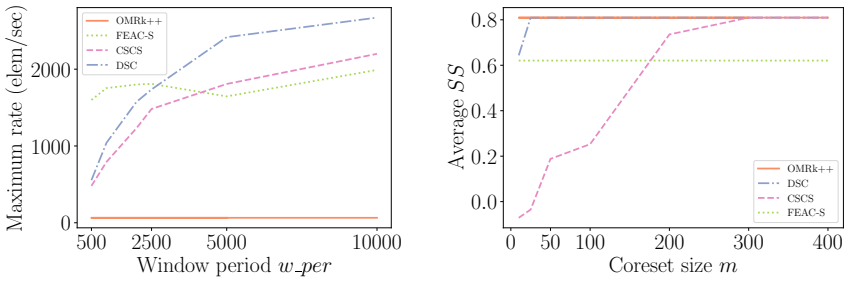
- *Sliding*: the elements of a cluster shift in the multidimensional space, causing the centroid to shift as well.
- *Fading out/in*: an existing cluster disappears or a new one appears.
- *Splitting*: the elements of a cluster grow in number/volume and become best captured by two or more clusters rather than one.
- *Merging*: the elements of two clusters shift closer to each other and become best captured by a single cluster.

Related works usually adopt basic update rules that can only partly capture the above-mentioned phenomenon. Applied to our context, they consist in i) associating newly created buckets with the cluster whose centroid is closest on d_{WJ} and recomputing the centroids; ii) deleting clusters that are left empty (because old buckets have expired and new buckets have been assigned with other clusters). These rules only capture the sliding and fading out of clusters.

In this work, these basic rules are extended with more advanced ones to capture the splitting, merging, and fading in phenomena. The challenge in defining these rules is to remain as faithful as possible to the behavior of the original k-means. It is known that k-means tends to create spherical and homogeneous clusters; in particular, the Simplified Silhouette (i.e., the objective function of OMRk++ in choosing the best k) aims at minimizing cluster scattering (i.e., having well-grouped elements) and maximizing cluster separation (i.e., clusters distant from each other). By controlling these properties, the following rules are implemented.

1. **Splitting rule.** The splitting of a cluster is determined by analyzing its *scattering*, i.e., the average distance between its summaries and its centroid. When a cluster exhibits unusually high scattering compared to others, it is split into two or more clusters by running OMRk++ on its elements.
2. **Merging rule.** The merging of two clusters is guided by the *separation* between them, i.e., the distance between their centroids. Two clusters are merged if (i) their separation is unusually high compared to other couples of clusters, and (ii) when they overlap, i.e., when the distance between the centroids is less than the sum of their radii (measured as the scattering of the clusters).

Notice that the splitting rule implicitly captures the fading in phenomenon as well, thus completing the covering of all mentioned phenomena. While these rules adhere to the principle of k-means (and iteratively applied, as more split/merge operations may be necessary), they do not guarantee an improvement of the profile. Thus, we validate the resulting profile by checking if the SS has improved compared to before applying the rule; otherwise, the reclustering policy is triggered and the profile recomputed from scratch.



(a) (b)

Fig. 2: (a) Average execution time by different window periods (i.e., number of elements in the pane) for $m = 100$. (b) Average SS by different values for m (coreset size).

4 Experiments

The experimental evaluation is done to assess the effectiveness and efficiency of DSC. Tests are executed on a single machine with a RAM of 16GB and an i7-4790 CPU @3.60 GHz. The comparison is carried out against the reference baseline – which consists in running OMRk++ over the whole window of schemas (not summaries) at every iteration (i.e., for every new pane of data) – and the two closest state-of-the-art proposals for streaming k-means, i.e., CSCS [14] and FEAC-S [2]. To avoid excessive execution times, the maximum value of k for the baseline OMRk++ is set to 20. CSCS has been adapted to work in our context as follows: (i) to support variable k, the clustering of the coreset is done multiple times for different k as in OMRk++; (ii) since the variable domain of attributes breaks the compaction technique to define bucket identifiers, the schema elements in the window are retained and buckets redefined whenever the attribute domain changes. All algorithms have been implemented in Scala.

The evaluation is based on a generator simulating a stream of JSON schemas generated by multiple seeds (i.e., sources) with a given degree of variability: each seed generates schemas with around 10 attributes, chosen with probability 0.9 from a pool of exclusive or shared attributes, in order to create similar and slightly overlapped clusters. The generator works either in a static or evolutionary manner, where the latter implements mechanisms to simulate the phenomena discussed in Section 3.2; these mechanisms are gradually activated after a short period (approximately 5 to 10 panes), in order to test the incremental update procedure. The generated datasets are referred to as D_{static} (with 10 seeds), D_{fadein} (where the 10^{th} seed is turned on only later), $D_{fadeout}$ (where a seed is turned off), D_{slide} (where a seed produces schemas with progressively different attributes), D_{split} (where a seed progressively generates two separate patterns of schemas), and D_{merge} (where two seeds progressively generate schemas converging towards the same attributes). The default parameters for both the DSC and

Dataset	Measure	OMRk++	DSC	CSCS1 [14]	CSCS3 [14]	FEAC-S [2]
D_{static}	Time (s)	35.90	0.42	0.48	0.36	**0.13**
	SS	**0.81**	**0.81**	0.24	**0.81**	0.62
D_{fadein}	Time (s)	33.76	0.41	0.40	0.36	**0.12**
	SS	**0.81**	0.80	0.39	0.75	0.57
$D_{fadeout}$	Time (s)	32.08	0.38	0.44	0.31	**0.12**
	SS	**0.81**	**0.81**	0.23	0.78	0.46
D_{slide}	Time (s)	35.08	0.42	0.58	0.47	**0.13**
	SS	**0.80**	**0.80**	0.42	**0.80**	0.56
D_{split}	Time (s)	28.22	0.43	0.52	0.49	**0.13**
	SS	**0.80**	0.79	0.32	0.78	0.46
D_{merge}	Time (s)	35.74	0.44	0.54	0.47	**0.13**
	SS	**0.78**	**0.78**	0.30	0.76	0.47

Table 1: Algorithms' performances on the different datasets.

CSCS algorithms are: $m = 100$ (coreset size) and $l = 15$ (number of functions to compute identifiers).

The efficiency of the algorithms is measured on D_{static} by varying the number of panes per window, whose size is fixed to $w\_len = 10000$. In Figure 2a we show the maximum stream rate supported in terms of elements per second; it is calculated as the number of elements in the pane over the execution time in seconds, which intuitively indicates the maximum rate tolerated by the algorithm to keep up with the flow of data in the stream. The results in Figure 2a show that (i) DSC runs constantly faster than CSCS, thus it is able to support a higher stream rate; (ii) since both DSC and CSCS pre-aggregate elements into a coreset, they are able to scale better and to increase the maximum rate as the pane size increases; differently, FEAC-S directly inserts every element into the closest cluster, thus its execution time is linear with respect to the number of elements in the pane and its maximum rate remains constant; (iii) OMRk++ lies at the bottom at only 65 elem/sec.

The effectiveness of the algorithms is first tested on D_{static}; here, we measure the simplified silhouette (SS) of the obtained solution by varying the size of the coreset m on D_{static} (which affects only DSC and CSCS, thus the SS for OMRk++ and FEAC-S is constant). The length and period of the window are $w\_len = 2000$ are $w\_per = 200$. Figure 2b shows the averages of the values obtained in every pane iteration. Interestingly, DSC is able to converge to the baseline reference values with a much smaller coreset size than CSCS, indicating that the way the coreset is created and maintained is more accurate – whereas the quality of the result by FEAC-S is quite lower than the baseline's and DSC's.

Table 1 compares the algorithms over all datasets. The length and period of the window are $w\_len = 2000$ and $w\_per = 200$. As CSCS struggles in providing good results with $m = 100$, it is tested also with $m = 100$ (CSCS1) and $m = 300$ (CSCS3). The results show that DSC is able to return results adhering to the OMRk++ baseline in all datasets (it is not shown for space constraints, but changes in the number of clusters are all captured) under much better performances. DSC also proves better than CSCS, with great improvements on the effectiveness side obtained with comparable (if not better) execution times – especially notable in the datasets containing a change in the number of clus-

ters, which CSCS cannot capture. FEAC-S shows faster execution times, but the accuracy of its results is very distant from the baseline's and DSC's.

5 Conclusions

In this paper, a streaming approach to schema profiling has been presented. It proposes DSC; an approximate k-means algorithm with variable k to compute the profile, designed to pre-aggregate schemas and incrementally update the profile, thus minimizing the need to run the clustering algorithm. DSC extends state-of-the-art streaming algorithms by supporting the domain of schema attributes, which is variable and unknown, and fully capturing the evolution of clusters. Experiments demonstrate the effectiveness of DSC in providing an accurate profile and higher performances than the closest related works. Future work will be aimed to (i) further improve the efficiency by simplifying the coreset creation and maintenance phase, (ii) evaluate DSC on real-world datasets, to further validate its effectiveness and possibly extended the rules to capture cluster evolution, and (iii) adopt DSC in analytical applications to support users beyond the monitoring of the stream and actively aiding the formulation of queries.

References

1. Akidau, T., et al.: Streaming systems: the what, where, when, and how of large-scale data processing. O'Reilly Media, Inc. (2018)
2. de Andrade Silva, J., et al.: An evolutionary algorithm for clustering data streams with a variable number of clusters. Expert Syst. Appl. (2017)
3. Arthur, D., et al.: k-means++: the advantages of careful seeding. SIAM (2007)
4. Breve, B., et al.: Dependency visualization in data stream profiling. Big Data Res. (2021)
5. Du, M., et al.: Spell: Streaming parsing of system event logs. IEEE Computer Society (2016)
6. Emmi, L.A., et al.: Digital representation of smart agricultural environments for robot navigation. CEUR Workshop Proceedings (2022)
7. Gallinucci, E., et al.: Schema profiling of document-oriented databases. Inf. Syst. (2018)
8. Grefenstette, G.: Explorations in automatic thesaurus discovery (1994)
9. Kullback, S., et al.: On Information and Sufficiency. The Annals of Mathematical Statistics (1951)
10. Levandowsky, M., et al.: Distance between sets. Nature (1971)
11. Naldi, M.C., et al.: Comparison among methods for k estimation in k-means. IEEE Computer Society (2009)
12. Naumann, F.: Data profiling revisited. SIGMOD Rec. (2013)
13. Seyfi, M., et al.: H-DAC: discriminative associative classification in data streams. Soft Comput. (2023)
14. Youn, J., et al.: Efficient data stream clustering with sliding windows based on locality-sensitive hashing. IEEE Access (2018)
15. Zhang, T., et al.: BIRCH: an efficient data clustering method for very large databases. ACM Press (1996)
16. Zubaroğlu, A., et al.: Data stream clustering: a review. Artificial Intelligence Review (2021)