# RPS: A Generic Reservoir Patterns Sampler

Lamine Diop*, Marc Plantevit*, Arnaud Soulet†
*EPITA Research Laboratory (LRE), Le Kremlin-Bicetre, Paris FR-94276, France, firstname.lastname@epita.fr
†University of Tours, LIFAT, 3 Place Jean Jaurès, Blois 41029, France, arnaud.soulet@univ-tours.fr

***Abstract*—Efficient learning from streaming data is important for modern data analysis due to the continuous and rapid evolution of data streams. Despite significant advancements in stream pattern mining, challenges persist, particularly in managing complex data streams like sequential and weighted itemsets. While reservoir sampling serves as a fundamental method for randomly selecting fixed-size samples from data streams, its application to such complex patterns remains largely unexplored. In this study, we introduce an approach that harnesses a weighted reservoir to facilitate direct pattern sampling from streaming batch data, thus ensuring scalability and efficiency. We present a generic algorithm capable of addressing temporal biases and handling various pattern types, including sequential, weighted, and unweighted itemsets. Through comprehensive experiments conducted on real-world datasets, we evaluate the effectiveness of our method, showcasing its ability to construct accurate incremental online classifiers for sequential data.**

***Index Terms*—Reservoir sampling, Output pattern sampling**

## I. Introduction

Stream data mining is a subset of data mining, aiming to extract valuable knowledge, patterns, and insights from continuously flowing data streams [1]. Unlike static data, data streams consist of an unbounded, constant flow of information from diverse sources such as sensors, social media, etc. Efficient algorithms have facilitated real-time analytics such as anomaly detection, retail analysis [2], and drift detection [3].

Despite these successes, stream data mining, characterized by its continuous and rapidly changing nature, poses unique challenges for traditional data processing techniques. Reservoir sampling has emerged as a fundamental method for randomly selecting a fixed-size sample from data streams. Reservoir pattern sampling has been recently proposed [4] by adapting the reservoir sampling approach [5] for itemset only. However, these techniques may face limitations when handling large and complex structured data such as sequential itemsets [6] or weighted itemsets [7]. These limitations underscore the ongoing need for innovative approaches to address the evolving complexities of stream data mining.

To overcome these challenges, we introduce an extension of the multi-step pattern sampling technique [8]–[10] tailored for stream data. Unfortunately, applying multi-step pattern sampling in complex and structured data streams remains unexplored. By leveraging a weighted reservoir, our approach enables the direct sampling and maintenance of patterns from streaming batch data, offering scalability and efficiency. We present a generic algorithm capable of handling temporal biases and various pattern types, such as sequential, weighted and unweighted itemsets. We also show the usefulness of the sampled patterns by proposing online classifiers on stream sequential itemsets with many models that were not able to run with sequential data.

The structure of this paper is organized as follows: In Section II, we provide a review of related work concerning reservoir sampling techniques and multi-step pattern sampling methods. Section III presents the fundamental definitions and formal problem statement. Our proposed generic solution is detailed in Section IV. In Section V, we conduct an evaluation of our approach using real-world and benchmark datasets, comparing the accuracy of a sample-based online sequential data classifier with state-of-the-art methods. Finally, we conclude the paper and discuss future directions in Section VI.

## II. Related Work

This section presents the reservoir sampling in stream data and the local multi-step pattern sampling literature.

### A. *Reservoir sampling*

Reservoir sampling is a fundamental technique in computer science and statistics. It is used to address the problem of randomly selecting a fixed-size sample from a stream of data without knowing the total number of elements in advance [11] without lost of soundness, where traditional methods like sorting or storing all the data are impractical due to memory constraints [12]. Recently, it has been extended to output pattern sampling (*sample of patterns from the pattern language*) in stream itemsets [4] where each transaction or itemset $\gamma$ is spread into a set of patterns, $2^\gamma \setminus \emptyset$, without materialize it. After that, the set of patterns is scanned using a binary index operator to draw a pattern directly. However, the reservoir sampling technique proposed in [4] is not generic because the key idea which based on the binary index operator is not applicable with complex structure such as sequence [6] and quantitative data (weighted itemsets) [7].

### B. *Local multi-step output pattern sampling*

Multi-step pattern sampling [8] is the fastest among the techniques used in output space pattern sampling [13] to draw representative patterns directly from the database. Particularly efficient for sampling in local data, multi-step is widely regarded as the most efficient approach, especially following the preprocessing phase, which involves computing the normalization constant. This method has been successfully applied across different pattern languages, including itemset [8], numerical data [14], sequential data [9], and quantitative data [10]. The primary concept behind this technique is to draw a pattern directly from the database with a probability proportional to a given interestingness measure $m$. However, one of its most intricate limitations is the requirement to know the normalization constant, which can be time-consuming with very large databases or unfeasible with stream data [1].

In this paper, we demonstrate how to extend the multi-step pattern sampling technique to sample and maintain a set of patterns directly from stream data based on a weighted reservoir. We propose a generic algorithm capable of handling itemsets, sequential patterns, and high utility itemsets while incorporating norm-based utility to address the long-tail issue.

## III. Problem Statement

This section formalizes the problem of reservoir-based multi-step pattern sampling under norm-based utility measure. We first recall some preliminary definitions about structured patterns and stream data.

### A. Definitions and preliminaries

Let $\mathcal{I} = \{e_1, \ldots, e_N\}$ be a set of finite literals called items. An itemset $X$ is a non-empty subset of $\mathcal{I}$, i.e., $X \in 2^{\mathcal{I}} \setminus \emptyset$. The set of all itemsets in $\mathcal{I}$ is called the pattern language for itemset mining, denoted by $\mathcal{L}_{\mathrm{I}}$. An instance $\gamma = \langle X_1, \ldots, X_n \rangle$ defined over $\mathcal{I}$ is an ordered list of itemsets $X_i \in \mathcal{L}_{\mathcal{I}}$ $(1 \leq i \leq n, n \in \mathbb{N})$. $n$ is the size of the instance $\gamma$ denoted by $|\gamma|$. If $|\gamma| > 1$ then $\gamma$ is a sequence of itemsets and $\mathcal{L}_{\mathrm{S}}$ denote the universal set of all sequences defined over $\mathcal{I}$, otherwise $\gamma$ is an itemset also called a transaction denote by $\gamma = X_1$ for simplicity. A transaction can be weighted and the patterns mined from it are called high utility itemset (HUI) in general. The norm of an instance $\gamma$, denoted by $\|\gamma\|$, is the sum of the cardinality of all its itemsets, i.e., $\|\gamma\| = \sum_{i=1}^{n} |X_i|$. Finally, given a pattern language $\mathcal{L} \in \{\mathcal{L}_{\mathrm{I}}, \mathcal{L}_{\mathrm{S}}\}$, pattern $\varphi \in \mathcal{L}$ can be generally defined as follows:

**Definition 1** (Pattern)**.** *$\varphi = \langle X'_1, \ldots, X'_{n'} \rangle$ is a pattern or an generalization of an instance $\gamma = \langle X_1, \ldots, X_n \rangle$, denoted by $\varphi \preceq \gamma$, if there exists an index sequence $1 \leq i_1 < i_2 < \ldots < i_{n'} \leq n$ such that for all $j \in [1..n']$, one has $X'_j \subseteq X_{i_j}$.*

This definition is usually used in the context of sequential pattern mining, but we recall that an itemset is nothing else that a sequential pattern of length 1.

**Data stream and interestingness utility measures:** In general, we denote $\mathcal{L} \in \{\mathcal{L}_{\mathrm{I}}, \mathcal{L}_{\mathrm{S}}\}$ as a pattern language. A data stream is a sequence of batches with timestamps denoted as follows: $\mathcal{D} = \langle (t_1, \mathbf{B}_1), \ldots, (t_n, \mathbf{B}_n) \rangle$, such that $\mathbf{B}_j \subseteq \mathcal{L}$ for all $j \in [1..n]$ and $t_j < t_{j+1}$ for all $j \in [1..n-1]$, where a batch is a set of finite instances send at the same time, i.e.,

$$\mathbf{B}_j = \{\gamma_{j_1}, \cdots, \gamma_{j_{j'}} : (\gamma_{j_k} \in \mathcal{L})(\forall\ k \in [1..j'])\}.$$

In other words, a batch contains a set of instances that have equal temporal relevance. $\mathcal{L}(\mathcal{D})$ is the set of all patterns that can be mined from $\mathcal{D}$. In this paper, we consider the *Landmark window* time constraint, which provides a structured way to divide the data stream into manageable chunks called instances, and *damped window* which favors the recent instances. We also use other constraints and utility measures that can combine frequency and norm-based utility measures.

**Definition 2** (Frequency)**.** *Given a database $\mathcal{D}$ defined over a pattern language $\mathcal{L}$, the frequency of a pattern $\varphi \in \mathcal{L}$ denoted $freq(\varphi, \mathcal{D})$, is the number of instances that support. Formally, it is defined as follows:*

$$freq(\varphi, \mathcal{D}) = |\{\gamma \in \mathbf{B} : ((t, \mathbf{B}) \in \mathcal{D}) \wedge (\varphi \preceq \gamma)\}|.$$

In pattern mining, frequency is often associated with other interestingness measures to reveal meaningful insights. In this paper, we combine it with other measures, specifically norm-based utility measures [15], to identify truly interesting and actionable patterns. It is also possible and helpful to use norm-based utility measures in high utility itemset discovery.

**Definition 3** (Norm-based utility [15])**.** *A utility function $F_m$ is a norm-based utility if there exists a function $f_m : \mathbb{N} \rightarrow \mathbb{R}$ such that for every pattern $\varphi \in \mathcal{L}$, one has $F_m(\varphi) = f_m(\|\varphi\|)$.*

**Important remarks:** With weighted items, the utility is not norm-based because the weights $\omega(e, \gamma)$ of each item $e$ depend on the transaction $\gamma$ of the database in which it appears. In this case, each pattern $\varphi \subseteq \gamma$ has a utility within the transaction $\gamma$ defined by $U(\varphi, \gamma) = \sum_{e \in \varphi} \omega(e, \gamma)$. Therefore, with the language $\mathcal{L}_{\mathrm{I}}$, if the items are not weighted, we consider $U(\varphi, \gamma) = 1$ if $\varphi \subseteq \gamma$. It is also essential to note that, with sequential data defined over $\mathcal{L}_{\mathrm{S}}$, we have $U(\varphi, \gamma) = 1$ if $\varphi \subseteq \gamma$ since we do not deal with high utility sequential patterns mining. Obviously, for any pattern language $\mathcal{L} \in \{\mathcal{L}_{\mathrm{I}}, \mathcal{L}_{\mathrm{S}}\}$, we consider $U(\varphi, \gamma) = 0$ if $\varphi \not\subseteq \gamma$. Based on these remarks, we introduce the following definition:

**Definition 4** (Norm-based utility measure)**.** *Let $\gamma$ be an instance and $\varphi$ a pattern defined over $\mathcal{L}$. The norm-based utility measure, also said the interestingness utility measure of $\varphi$ within $\gamma$, denoted by $m(\varphi, \gamma)$, is defined as follows:*

$$m(\varphi, \gamma) = U(\varphi, \gamma) \times F_m(\varphi).$$

In general, we are interested by the utility of a pattern in the entire database that we call the pattern global utility.

**Definition 5** (Pattern Global Utility)**.** *Let $\mathcal{D}$ be a database defined over a pattern language $\mathcal{L}$ and $m$ an interestingness utility measure. The global utility of $\varphi$ in $\mathcal{D}$ is given by: $G_m(\varphi, \mathcal{D}) = \sum_{(t,\mathbf{B}) \in \mathcal{D}} \left( \sum_{\gamma \in \mathbf{B}} m(\varphi, \gamma) \right)$.*

In stream data under temporal biases, the utility of a pattern $\varphi$ inserted at time $t_j$ can be different to its utility at time $t_n$, with $n > j$. It depends on what the user really needs to favor, recent or all patterns, by weighting each pattern with a temporal bias. Therefore, we introduce a generic damping function defined as follows:

**Definition 6** (Damping function $\nabla_\varepsilon(t_n, t_j)$)**.** *The temporal bias is used when a user needs to favor the recent patterns or not. It is based on a damping factor $\varepsilon \in [0, 1]$. At time $t_n$, the* *temporal bias of each visited instance $\gamma_j$ at time $t_j \leq t_n$ are formally updated as follows: $\nabla_\varepsilon(t_n, t_j) = e^{-(t_n - t_j)\times\varepsilon}$.*

We can see that if $\varepsilon = 0$, then $\nabla_\varepsilon(t_n, t_j) = 1$, which corresponds to the landmark window.

To take account these temporal biases in our approach, we define the pattern global utility under temporal bias as follow:

**Definition 7** (Pattern Global Utility under temporal bias)**.** *Let $\mathcal{D} = \langle(t_1, \mathbf{B}_1), \ldots, (t_n, \mathbf{B}_n)\rangle$ be a stream data defined over a pattern language $\mathcal{L}$, $m$ be an interestingness utility measure and $\varepsilon \in [0, 1]$ a damping factor. At time $t_n$, the global utility of any pattern $\varphi$ inserted into the reservoir at time $t$ and that still appears into $\mathcal{S}$ is given by: $G^\varepsilon_m(\varphi, \mathcal{D}) = \sum_{(t_i,\mathbf{B}_i)\in\mathcal{D}} \left(\left(\sum_{\gamma_{i_j}\in\mathbf{B}_i} m(\varphi, \gamma_{i_j})\right) \times \nabla_\varepsilon(t_n, t_i)\right)$.*

**Example 1.** *Table I and Table II present two toy datasets respectively for sequential and weighted itemsets. They also give some measures with temporal biases and utility measures. The damping factor is set to $\varepsilon \in \{0, 0.1\}$. For Table II, $\{A, B, C\}$, $\{2, 1.5, 2\}$ means that the items A, B, and C have weights $2$, $1.5$ and $2$ respectively in the instance $\gamma_1$.*

TABLE I: Stream batches of sequential itemsets

| $\downarrow$ time | id | Sequential itemsets | $U(\langle\{A\}\{C\}\rangle, \gamma_i)$ |
|---|---|---|---|
| $(t_1, \mathbf{B}_1)$ | $\gamma_1$ | $\langle\{\underline{A}\}\{B\}\{A, \underline{C}\}\{B\}\rangle$ | 1 |
| | $\gamma_2$ | $\langle\{\underline{A}, B, C\}\{\underline{C}\}\{A, C\}\rangle$ | 1 |
| $(t_2, \mathbf{B}_2)$ | $\gamma_3$ | $\langle\{B\}\{A, C\}\{A\}\rangle$ | 0 |

| | |
|---|---|
| $G_{freq}(\langle\{A\}\{C\}\rangle, \mathcal{D})$ | $1 + 1 + 0 = 2$ |
| $G^0_{area}(\langle\{A\}\{C\}\rangle, \mathcal{D})$ | $(1 \times 2 + 1 \times 2) \times 1 + (0 \times 2) \times 1 = 4$ |
| $G^{0.1}_{freq}(\langle\{A\}\{C\}\rangle, \mathcal{D})$ | $(4) \cdot e^{-(2-1)\cdot 0.1} + (0) \cdot e^{-(2-2)\cdot 0.1} \approx 3.6$ |

TABLE II: Stream batches of weighted itemsets

| $\downarrow$ time | id | Weighted itemsets | $U(\{B, C\}, \gamma_i)$ |
|---|---|---|---|
| $(t_1, \mathbf{B}_1)$ | $\gamma_1$ | $\{A, B, C\}$, $\{2, 1.5, 2\}$ | $1.5 + 2 = 3.5$ |
| $(t_2, \mathbf{B}_2)$ | $\gamma_2$ | $\{A, C\}$, $\{3, 3\}$ | 0 |
| | $\gamma_3$ | $\{B, C, D, E\}$, $\{2, 1, 2, 1\}$ | $2 + 1 = 3$ |

| | |
|---|---|
| $G_{hui}(\{B, C\}, \mathcal{D})$ | $3.5 + 0 + 3 = 6.5$ |
| $G^0_{haui}(\{B, C\}, \mathcal{D})$ | $(3.5 \times \frac{1}{2} + 0 \times \frac{1}{2}) \times 1 + (3 \times \frac{1}{2}) \times 1 = \frac{6.5}{2}$ |
| $G^{0.1}_{hui}(\{B, C\}, \mathcal{D})$ | $(3.5) \cdot e^{-(2-1)\cdot 0.1} + (3) \cdot e^{-(2-2)\cdot 0.1} \approx 6.2$ |

### *B. Reservoir-based multi-step pattern sampling problem*

A reservoir-based pattern sampling aims to randomly maintains a sample of patterns proportionally to a given utility measure. For instance, in the case of area measure without constraints, we have $area(\varphi, \gamma) = \|\varphi\|$, and the global utility under landmark temporal bias of $\varphi$ in $\mathcal{D}$ is given by $G^0_m(\varphi, \mathcal{D}) = \sum_{(t,\mathbf{B})\in\mathcal{D}}(\sum_{\gamma\in\mathbf{B}} area(\varphi, \gamma)) \times \nabla_0(t_n, t) = freq(\varphi, \mathcal{D}) \times \|\varphi\|$. This measure is applicable to both types of pattern languages, $\mathcal{L}_\mathrm{I}$ and $\mathcal{L}_\mathrm{S}$.

**Problem Definition** (Reservoir pattern sampling)**.** *Given a stream of data $\mathcal{D} = \langle(t_1, \mathbf{B}_1), \ldots, (t_n, \mathbf{B}_n)\rangle$ defined over a pattern language $\mathcal{L} \in \{\mathcal{L}_I, \mathcal{L}_S\}$, a damping factor $\varepsilon \in [0, 1]$, and an interestingness utility measure $m$, our goal is to maintain a reservoir of size $k$ patterns $[\varphi_1, \ldots, \varphi_k]$ from $\mathcal{L}$. Each pattern $\varphi_j$ should be drawn with a probability proportional to its weighted utility in the database $\mathcal{D}$, defined as: $\mathbb{P}(\varphi_j, \mathcal{D}) = \frac{G^\varepsilon_m(\varphi_j, \mathcal{D})}{\sum_{\varphi\in\mathcal{L}(\mathcal{D})} G^\varepsilon_m(\varphi, \mathcal{D})}$.*

## IV. Reservoir-based Multi-Step Pattern Sampling

A multi-step pattern sampling approach is a sampling technique with replacement. Therefore, in this paper, we focus on weighted reservoir sampling with replacement. We detail below the three steps that our approach should follows.

*a)* **Step 1.** *Batch acceptance probability:* Let us assume that all the positions of the reservoir are already occupied by a pattern from past batches $\langle(t_1, \mathbf{B}_1), \ldots, (t_n, \mathbf{B}_{j-1})\rangle$ and that all batch weights $\omega_m(\mathbf{B}_i) = \sum_{\gamma_{i_{j'}}\in\mathbf{B}_i} \sum_{\varphi\preceq\gamma_{i_{j'}}} m(\varphi, \gamma_{i_{j'}})$, with $i \in [1..j-1]$ are feasible. For any batch $\mathbf{B}_j$ with a weight $\omega_m(\mathbf{B}_j)$, we compute the probability acceptance that of one of its patterns from $\mathcal{L}(\mathbf{B}_j)$ be inserted into the reservoir $p_j$. Based on [5], we have $p_j = \frac{\omega_m(\mathbf{B}_j)}{\sum_{i=1}^{j} \omega_m(\mathbf{B}_i)\times\nabla_\varepsilon(t_j, t_i)}$. Therefore, a pattern drawn proportionally to its weight in $\mathbf{B}_j$ can be inserted at a position of the reservoir uniformly drawn. This uniform replacement has been already used by A-Chao [16] for weighted reservoir sampling in data stream.

However, it is evident that computing the acceptance probability by iteratively summing the weights of the visited batches is infeasible because we do not store the past batches. Therefore, we employ a memory-less computing technique to adapt the normalization constant that avoids storing information for each batch received.

**Property 1.** *Let $\mathbf{Z}_{i-1}$ be the normalization constant for the $i-1$ first batches of the data stream under the damped window and the norm-based utility measure $m$, with $\mathbf{Z}_0 = 0$. The probability $p$ to draw a pattern from the next batch $\mathbf{B}_i$ under the damped window can be computed as follows:*
*$p = \frac{\omega_m(\mathbf{B}_i)\times e^{\varepsilon\times t_i}}{\mathbf{Z}_i}$, with $\mathbf{Z}_i = \mathbf{Z}_{i-1} + \omega_m(\mathbf{B}_i) \times e^{\varepsilon\times t_i}$.*

After the acceptance of batch $\mathbf{B}_j$ with a probability of $p_j$ in Step 1, we know that at least one of its patterns should be inserted into the reservoir. But, there is also the possibility of other patterns from the set being inserted at various positions within the reservoir. This drives the rationale behind the second step of our approach.

*b)* **Step 2.** *Number of patterns to draw from an accepted batch:* Let $\mathcal{S}$ be a reservoir of size $k$ where we need to store a sample from a population of finite size. In this case, each pattern can be selected up to $n_r \leq k$ times in the sample. To achieve this goal, we process $k$ copies of $\mathcal{L}(\mathbf{B}_j)$ such in each of them, a pattern is drawn proportionally to its weight in $\mathbf{B}_j$. Interestingly, using $k$ copies of $\mathcal{L}(\mathbf{B}_j)$ with a probability $p_j$ corresponds to simulating $k$ independents Bernoulli trials with a probability $p_j$. However, we note that computing a probability acceptance for each position is time consuming. Therefore, to skip computing an acceptance rate for each position, we use the Cumulative Binomial Probability Distribution defined as follows (CBPD). Interestingly, the CBPD can be more efficiently computed by using the incomplete beta function (IBF), $\mathbf{I}_x(a, b)$ [17]. By leveraging the IBF, we can efficiently handle complex probability calculations without explicit summation of individual probabilities minimizing computational load and numerical errors.

**Definition 8** (Incomplete beta function (IBF) [18])**.** *The incomplete beta function $I_x(a,b)$ is defined as follows: $\mathbf{I}_x(a,b) = \frac{1}{B(a,b)} \int_0^x t^{a-1}(1-t)^{b-1}\,dt$ where a and b are positive real numbers (parameters of the beta distribution); $B(a,b)$ is the beta function, defined as: $B(a,b) = \int_0^1 t^{a-1}(1-t)^{b-1}\,dt$ and $x$ is a real number in the range $[0,1]$.*

**Property 2** (From CBPD to IBF)**.** *Let $k$ be the total number of trials, $n_r \leq k$ the minimum number of times the event must occur, and $p$ the probability of the event occurring in a single trial. The Cumulative Binomial Probability Distribution can be computed as follows:*

$$\mathbb{P}(n_r, k) \equiv \sum_{j=n_r}^{k} \binom{k}{j} p^j (1-p)^{k-j} = \mathbf{I}_p(n_r, k-n_r+1). \quad (1)$$

**Definition 9** (Inverse Incomplete Beta Function)**.** *The Inverse Incomplete Beta Function allows for the approximation of the number of successful trials $n_r$ out of $k$ trials that matches the CBPD with a given probability $x \in [0,1]$ as follows:*

$$n_r = \arg_{n_r}[\mathbf{I}_p(n_r, k-n_r+1) = x].$$

Now we are going to show how to draw a pattern proportionally to its interest from an accepted batch.

*c) Step* **3.** *First pattern occurrence sampling from a batch:* The main goal of processing a copy of an acceptance batch is finally to draw a pattern. However, the complexity of sampling a pattern from a batch depends on the pattern language. With sequential itemsets, a pattern can have multiple occurrences within a sequence [9] which is not the case with weighted/unweighted itemsets. Since we propose a generic approach dealing with sequential itemsets, then we adapt the first occurrence definition previously introduced in [9].

**Definition 10** (First occurrence)**.** *Given an instance $\gamma$, let $o_1$ and $o_2$ be two occurrences of a pattern $\varphi$ within $\gamma$, whose signatures are $\langle i_1^1, i_2^1, \ldots, i_N^1\rangle$ and $\langle i_1^2, i_2^2, \ldots, i_N^2\rangle$ respectively. $o_1$ is less than $o_2$, denoted by $o_1 < o_2$, if there exists an index $k \in [1..N]$ such that for all $j \in [1..k-1]$, one has $i_j^1 = i_j^2$, and $i_k^1 < i_k^2$. Finally, the first occurrence of $\varphi$ in $\gamma$ its smallest occurrence with respect to the order defined previously.*

**Example 2.** *For instance, $\varphi = \langle\{A\}\{C\}\rangle$ has two occurrences $o_1$ and $o_2$ in $\gamma_2 = \langle\{A,B,C\}\{C\}\{A,C\}\rangle$ with signatures $\langle 1,2\rangle$ and $\langle 1,3\rangle$ respectively. But $o_1$ is the first occurrence because $o_1 < o_2$ since $2 < 3$.*

Based on Definition 10, we then propose a generic sampler operator named $\text{Sample}_{n_r}(\mathcal{L}, \mathbf{B}, m)$. In fact, running $n_r$ times the operator $\text{Sample}_1(\mathcal{L}, \mathbf{B}, m)$ in the same batch $\mathbf{B}$ in order to get a first pattern occurrence for each realization is equivalent to running $\text{Sample}_{n_r}(\mathcal{L}, \mathbf{B}, m)$ once because $\bigcup_{i=1}^{n_r}\{\varphi_i \sim m(\mathcal{L}, \mathbf{B})\} \equiv \text{Sample}_{n_r}(\mathcal{L}, \mathbf{B}, m)$. We are going to present our generic algorithm using weighted reservoir sampling with replacement.

## A. A Generic Reservoir-based Three-Step Pattern Sampling

We first give a high-level description of RPS described in Algorithm 1. It takes a data stream $\mathcal{D}$, a utility measure $m$, the desired reservoir size $k$, and a damping factor $\varepsilon \in [0,1]$. First, for each batch $\mathbf{B}_i$ appearing at timestamp $t_i$, the acceptance probability $p$ that a pattern from $\mathcal{L}(\mathbf{B}_i)$ replaces a pattern inserted at $t_{i'}$, with $t_{i'} < t_i$ is computed (lines 3-4). If $\mathbf{B}_i$ is accepted (line 5), which correspond to a success trial, then the number of additional success trials out of the rest of the reservoir size $k-1$ that a pattern of $\mathcal{L}(\mathbf{B}_i)$ should be inserted (line 6) is deduced based on the inverse IBF (Definition 9). Lines 7 to 10 allow to draw $n_r$ patterns with replacement where each draw corresponds to an inserted pattern. At time $t_n$, RPS maintains a reservoir of $k$ patterns where each pattern $\varphi$ is selected with a probability proportional $G_m^{\varepsilon}(\varphi, \mathcal{D})$.

**Algorithm 1** RPS: A Generic Stream pattern sampler

```
    Input: A data stream $\mathcal{D}$, a utility measure $m$, a damping
    factor $\varepsilon \in [0,1]$, and the desired reservoir size $k$
    Output at time $t_n$: A sample $\mathcal{S}$ of $k$ patterns drawn in
    $\mathcal{L}(\mathcal{D} = \langle(t_1, \mathbf{B}_1), \ldots, (t_n, \mathbf{B}_n)\rangle)$ based on $m$ and $\varepsilon$
 1: $\mathcal{S} \leftarrow \emptyset$; $\mathbf{Z}_0 = 0$
 2: while $(t_i, \mathbf{B}_i)$ is from $\mathcal{D}$ do
        //Batch acceptance probability
 3:     $\mathbf{Z}_i = \mathbf{Z}_{i-1} + \omega_m(\mathbf{B}_i) \times e^{\varepsilon \times t_i}$
 4:     $p \leftarrow \frac{\omega_m(\mathbf{B}_i) \times e^{\varepsilon \times t_i}}{\mathbf{Z}_i}$ and $x \leftarrow random(0,1)$
 5:     if $p > x$ then
            //Number of realisations
 6:         $n_r \leftarrow 1 + \underset{n_r}{\arg}[\mathbf{I}_p(n_r, k-n_r) = x]$  ▷ Definition 9
            //Patterns selection
 7:         $E \leftarrow getPatternsToRemove(\mathcal{S}, n_r)$
 8:         $\mathcal{S} \leftarrow \mathcal{S} \setminus \{\mathcal{S}[j] :$ for $j \in E\}$
 9:         for $\varphi_j \in \text{Sample}_{n_r}(\mathcal{L}, \mathbf{B}_i, m)$ do
10:             $\mathcal{S} \leftarrow \mathcal{S} \cup \{(t_i, \varphi_j)\}$
```

Regarding the $getPatternsToRemove$, it returns $n_r$ distinct indexes uniformly drawn from $[1..|\mathcal{S}|]$. By relying on [19], all patterns in the reservoir have an equal probability of being replaced by one of the $n_r$ patterns drawn from the current batch by the sampler $\text{Sample}_{n_r}(\mathcal{L}, \mathbf{B}_i, m)$.

## B. Theoretical analysis of the method

We study now the soundness and the complexity of RPS.

The following properties state that RPS returns an exact sample of patterns under temporal bias with norm-based utility measure.

**Property 3** (Batch acceptance probability)**.** *Given a stream data $\mathcal{D}$ defined over a pattern language $\mathcal{L}$, $\mathcal{D} = \langle(t_1, \mathbf{B}_1), \ldots, (t_n, \mathbf{B}_n)\rangle$, $m$ be a norm-based utility measure, $\varepsilon \in [0,1]$ a damping factor, and $k$ the size of the reservoir $\mathcal{S}$. After observing $(t_n, \mathbf{B}_n)$, the probability that a pattern of batch $\mathbf{B}_i$, inserted at time $t_i$ at the jth position of the reservoir, $j \in [1..k]$, stays in $\mathcal{S}$, denoted $\mathbb{P}(\mathbf{B}_i \triangleright \mathcal{S}[j] | t_n)$, is given by: $\mathbb{P}(\mathbf{B}_i \triangleright \mathcal{S}[j] | t_n) = \frac{\omega_m(\mathbf{B}_i) \times \nabla_\varepsilon(t_n, t_i)}{\sum_{i' \leq n} \omega_m(\mathbf{B}_{i'}) \times \nabla_\varepsilon(t_n, t_{i'})}$.*

Based on these properties, we can proof Property 4.

**Property 4** (Soundness)**.** *Let $\mathcal{D} = \langle(t_1, \mathbf{B}_1), \ldots, (t_n, \mathbf{B}_n)\rangle$ be a stream data defined over a pattern language $\mathcal{L}$, $m$ be a* *norm-based utility measure, $k$ the size of the reservoir $\mathcal{S}$ and $\varepsilon \in [0,1]$ a damping factor. After observing $(t_n, \mathbf{B}_n)$,* RPS *returns a sample of $k$ patterns $\varphi_1, \ldots, \varphi_k$ where each pattern $\varphi_j = \mathcal{S}[j]$ is drawn with a probability equals to: $\mathbb{P}(\varphi_j = \mathcal{S}[j], \mathcal{D}) = \frac{G_m^{\varepsilon}(\varphi_j, \mathcal{D})}{\sum_{\varphi \in \mathcal{L}} G_m^{\varepsilon}(\varphi_j, \mathcal{D})}$.*

## V. Experimental Study

We evaluate the efficiency of RPS and the interest of the sampled patterns. More precisely, Section V-A focuses on the speed of RPS with different batches and reservoir sizes as well as the maximal norm constraint. All experiments are performed on a 2.71 GHz 2 Core CPU with 12 GB of RAM and the prototype of our method is implemented in Python. The minimum norm constraint is set to $\mu = 1$ to avoid the empty set. Due to space limitations, we focus our experiments on sequential itemsets. In this case, we use the exponential decay [15] with $\alpha$ =0.001 and maximal norm constraint $M = 10$. The databases are from SPMF[1] and Ana[2].

### A. *Evaluation of* RPS *speed on sequential data*

Table III shows the behavior of RPS on different sequential databases as the reservoir size increases with different batch sizes. The experiments are repeated 5 times, and we can notice that the standard deviations are tiny. We see that the reservoir size has a slight impact on the execution time. Naturally, the batch size increases the execution time per batch. We also see that the damping factor ($\varepsilon$) impacts the execution time; the higher it is, the higher the execution time due to numerous insertions. In all cases, we notice that RPS uses a reasonable execution time even with larger database such as `cade` ($\approx 1,000s$) while others take less than $200s$.

### B. *Accuracy of Sampling-Based Online Classification*

In our framework, the learning phase precedes the prediction one. A learning-duration defines the length of the interval of timestamps in which our models learn based on the reservoir content. We also have the prediction-duration, which is the interval length for predicting only. As done in [9], we represent each sequence $\gamma$ in a batch $\mathbf{B}$ as a tuple of $k+1$ values, where $d[j] = 1$ if $\mathcal{S}[j] \preceq \gamma$ (0 otherwise) for $j \in [1..k]$, and $d[k+1] = c$, where $c$ is the class label of the sequence $\gamma$ and $k$ is the reservoir size. For the global accuracy, we use the average accuracy AvgAcc for an incremental online classifier over multiple timestamps $t_0, \ldots, t_{n-1}$, with $\text{Acc}(t_i)$ is the accuracy at timestamp $t_i$, as: $\text{AvgAcc} = \frac{1}{n}\sum_{i=0}^{n-1} \text{Acc}(t_i)$. For detecting deteriorating accuracy, we use the KSWIN (alpha=0.1) [20] with default parameter settings.

*a) Impact of the* RPS *parameters on the accuracy:* We first present the behavior of RPS-based online classifiers with different parameter changes, such as the sample size, the reservoir size, the learning duration, and the predict duration, by considering the `Books` database. Figure 2 shows that many RPS-based models improve their performance after each learning interval. Additionally, the maintained reservoir patterns are representative enough to provide good accuracies for long-term predictions, thereby avoiding accuracy drift.

*b) Accuracy comparison with cheater classifiers:* This section evaluates our approach by comparing it with cheater classifiers such as *Dumb classifier (strategy='most_frequent')*, *LogisticRegression (max_iter=1,000)*, *KNN (k=10)*, *Centroid (Normalized Sum)*, *Naive Bayes (MultinomialNB)*, and *SVM (Linear Kernel)*. These classifiers are our references since they have access to all the training data (50%), while RPS uses batches that appear during the learning intervals. This means that our goal is not to outperform them but to come closer to their performance as seen in Figure 1. However, RPS is always better than *Dumb classifier* which is a naturally lower baseline while demonstrating strong competitiveness.

## VI. Conclusion

We introduced RPS, a novel reservoir pattern sampling approach for complex structured data in streams, such as sequential and weighted itemsets. Our proposed method employs a multi-step technique that leverages the inverse incomplete Beta function and efficient computation of the normalization constant, resulting in a fast and effective pattern sampling approach. Our extensive experiments demonstrate the robustness and versatility of RPS. Notably, we adapted several classification models for online sequential data classification with new labels, showing that sampled patterns significantly enhance the accuracy of online classifiers, achieving performance comparable to offline baselines.

Future work will focus on extending RPS to graph streams, further broadening the applicability and impact of our research. An extended version of this paper is available at https://arxiv.org/abs/2411.00074.

[1]https://www.philippe-fournier-viger.com/spmf/index.php

[2]https://ana.cachopo.org/datasets-for-single-label-text-categorization

## References

[1] C. C. Aggarwal, P. S. Yu, J. Han, and J. Wang, "A framework for clustering evolving data streams," in *Proc. 2003 VLDB*, 2003, pp. 81–92.

[2] T. Toliopoulos, A. Gounaris, K. Tsichlas, A. Papadopoulos, and S. Sampaio, "Continuous outlier mining of streaming data in flink," *Information Systems*, vol. 93, p. 101569, 2020.

[3] Z. Liu and Chaozh, "Hypercalm sketch: One-pass mining periodic batches in data streams," in *IEEE ICDE*, 2023, pp. 14–26.

[4] A. Giacometti and A. Soulet, "Reservoir pattern sampling in data streams," in *ECMLPKDD 2021*, 2021, pp. 337–352.

[5] P. S. Efraimidis and P. G. Spirakis, "Weighted random sampling with a reservoir," *Inf. Process. Lett.*, vol. 97, no. 5, pp. 181–185, 2006.

[6] R. Srikant and R. Agrawal, "Mining sequential patterns: Generalizations and performance improvements," in *Advances in Database Technology - EDBT'96*, vol. 1057. Springer, 1996, pp. 3–17.

[7] V. S. Tseng, C. Wu, P. Fournier-Viger, and P. S. Yu, "Efficient algorithms for mining top-k high utility itemsets," *IEEE TKDE*, vol. 28, no. 1, pp. 54–67, Jan 2016.

[8] M. Boley, C. Lucchese, D. Paurat, and T. Gärtner, "Direct local pattern sampling by efficient two-step random procedures," in *KDD*, 2011, p. 582–590.

[9] L. Diop, C. T. Diop, A. Giacometti, D. Li Haoyuan, and A. Soulet, "Sequential Pattern Sampling with Norm Constraints," in *IEEE ICDM (ICDM)*, Singapore, Nov. 2018.

[10] L. Diop, "High average-utility itemset sampling under length constraints," in *PAKDD*, 2022, p. 134–148.

[11] J. S. Vitter, "Random sampling with a reservoir," *ACM Trans. Math. Softw.*, vol. 11, no. 1, p. 37–57, mar 1985.

[12] G. Cormode, S. Muthukrishnan, K. Yi, and Q. Zhang, "Optimal sampling from distributed streams," in *PODS*, 2010, p. 77–86.

[13] M. Al Hasan and M. J. Zaki, "Output space sampling for graph patterns," *Proc. VLDB Endow.*, vol. 2, no. 1, p. 730–741, aug 2009.

TABLE III: Average execution time per batch (in seconds) with different values of the damping factor ($\varepsilon \in \{0.0, 0.1, 0.5\}$), the batch size (in $\{1000, 1500, 2000\}$) and the reservoir size ($k \in \{1000, 3000\}$)

| Database | Batch size 1000 | | | Batch size 1500 | | | Batch size 2000 | | |
|---|---|---|---|---|---|---|---|---|---|
| | $\varepsilon = 0.0$ | $\varepsilon = 0.1$ | $\varepsilon = 0.5$ | $\varepsilon = 0.0$ | $\varepsilon = 0.1$ | $\varepsilon = 0.5$ | $\varepsilon = 0.0$ | $\varepsilon = 0.1$ | $\varepsilon = 0.5$ |
| Reservoir size $k$ =1000 | | | | | | | | | |
| `Books` | 0.36 ± 0.00 | 0.36 ± 0.00 | 0.36 ± 0.00 | 0.55 ± 0.01 | 0.56 ± 0.01 | 0.56 ± 0.01 | 0.70 ± 0.00 | 0.71 ± 0.00 | 0.72 ± 0.00 |
| `Reuters8` | 1.64 ± 0.01 | 1.65 ± 0.01 | 1.66 ± 0.03 | 2.57 ± 0.02 | 2.59 ± 0.02 | 2.60 ± 0.05 | 3.31 ± 0.01 | 3.34 ± 0.07 | 3.33 ± 0.06 |
| `Reuters52` | 1.79 ± 0.05 | 1.78 ± 0.01 | 1.79 ± 0.01 | 2.79 ± 0.01 | 2.80 ± 0.02 | 2.79 ± 0.02 | 3.52 ± 0.05 | 3.58 ± 0.01 | 3.59 ± 0.02 |
| `cade` | 22.14 ± 0.12 | 22.26 ± 0.06 | 22.89 ± 0.24 | 34.84 ± 0.09 | 34.82 ± 0.18 | 35.73 ± 0.24 | 44.39 ± 0.14 | 44.42 ± 0.25 | 45.24 ± 0.41 |
| `webkb` | 18.14 ± 1.46 | 17.65 ± 1.87 | 19.18 ± 0.15 | 29.51 ± 0.27 | 28.52 ± 2.44 | 29.94 ± 0.37 | 38.28 ± 0.17 | 38.18 ± 0.24 | 38.26 ± 0.33 |
| Reservoir size $k$ =3000 | | | | | | | | | |
| `Books` | 0.35 ± 0.00 | 0.36 ± 0.00 | 0.37 ± 0.00 | 0.53 ± 0.00 | 0.54 ± 0.00 | 0.55 ± 0.00 | 0.70 ± 0.00 | 0.72 ± 0.00 | 0.78 ± 0.02 |
| `Reuters8` | 1.67 ± 0.01 | 1.68 ± 0.01 | 1.69 ± 0.04 | 2.52 ± 0.02 | 2.55 ± 0.06 | 2.52 ± 0.03 | 3.35 ± 0.03 | 3.65 ± 0.06 | 3.48 ± 0.10 |
| `Reuters52` | 1.81 ± 0.03 | 1.80 ± 0.04 | 1.83 ± 0.02 | 2.73 ± 0.05 | 2.72 ± 0.04 | 2.70 ± 0.03 | 3.61 ± 0.02 | 3.62 ± 0.05 | 3.95 ± 0.22 |
| `cade` | 22.56 ± 0.15 | 22.84 ± 0.27 | 25.36 ± 0.99 | 34.05 ± 0.29 | 34.05 ± 0.34 | 36.36 ± 0.99 | 45.37 ± 0.54 | 48.46 ± 1.08 | 53.91 ± 5.66 |
| `webkb` | 26.51 ± 0.08 | 22.29 ± 6.22 | 27.43 ± 0.11 | 35.88 ± 7.20 | 36.37 ± 7.33 | 40.71 ± 0.12 | 54.44 ± 0.59 | 55.92 ± 0.59 | 56.99 ± 2.96 |

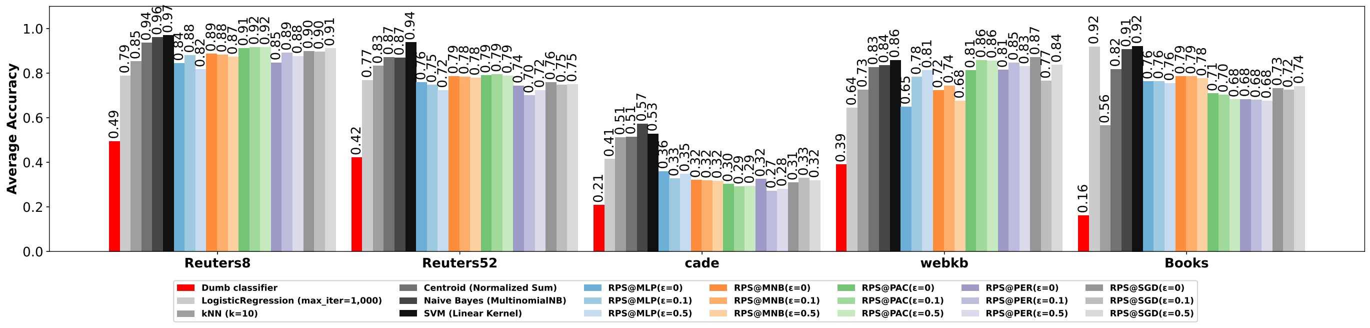

Fig. 1: Comparison between RPS-based classifiers (with reservoir size $k$ =10,000; batch size=1,000; learning duration=2 time-units, predict duration=52 time-units) vs cheater classifiers (with 50% train and 50% test)

TABLE IV: Parameters for our adapted models

| Model | Parameters |
|---|---|
| MNB [21] | alpha=0.0001 |
| PER [22] | max_iter=10000, tol=1e-3 |
| PAC [23] | max_iter=1000, tol=1e-3, loss='hinge' |
| MLP [24] | hidden_layer_sizes=(500,100,), activation='identity', solver='adam', max_iter=1000, warm_start=False, random_state=42 |
| SGD [25] | loss='hinge', penalty='l1', alpha=0.0001, max_iter=1000, tol=1e-3 |

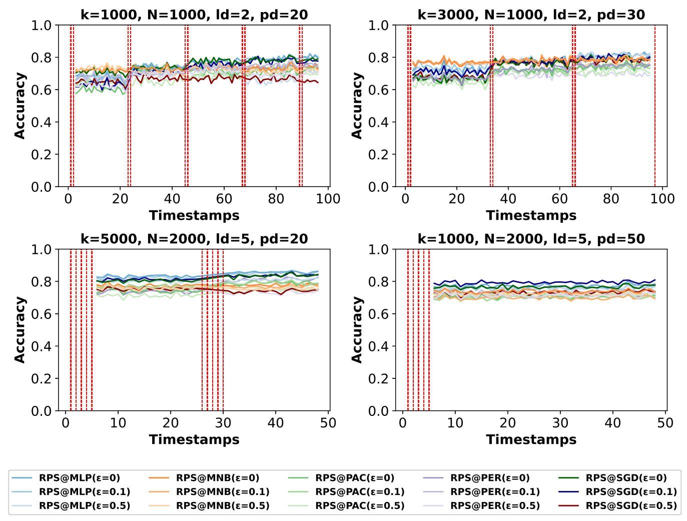

Fig. 2: Evolution of the accuracy per batch with different parameters on `Books`. Learning timestamps are in red. $k$: reservoir size, $N$: batch size, $ld$: learning duration, $pd$: predict duration

[14] A. Giacometti and A. Soulet, "Dense neighborhood pattern sampling in numerical data," in *Proc. of SDM 2018*, 2018, pp. 756–764.
[15] L. Diop, C. T. Diop, A. Giacometti, D. Li, and A. Soulet, "Sequential pattern sampling with norm-based utility," *Knowledge and Information Systems*, Oct 2019.
[16] M. T. CHAO, "A general purpose unequal probability sampling plan," *Biometrika*, vol. 69, no. 3, pp. 653–656, 12 1982.
[17] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, 1972.
[18] A. R. Didonato and M. P. Jarnagin, "The efficient calculation of the incomplete beta-function ratio for half-integer values of the parameters $a, b$," *Mathematics of Computation*, vol. 21, pp. 652–662, 1967.
[19] P. S. Efraimidis, *Weighted Random Sampling over Data Streams*. Cham: Springer International Publishing, 2015, pp. 183–195.
[20] J. Montiel, J. Read, A. Bifet, and T. Abdessalem, "Scikit-multiflow: A multi-output streaming framework," *Journal of Machine Learning Research*, vol. 19, no. 72, pp. 1–5, 2018.
[21] D. D. Lewis and W. A. Gale, "A sequential algorithm for training text classifiers," in *SIGIR '94*, B. W. Croft and C. J. van Rijsbergen, Eds. London: Springer London, 1994, pp. 3–12.
[22] F. Rosenblatt, "The perceptron: a probabilistic model for information storage and organization in the brain." *Psychological review*, vol. 65 6, pp. 386–408, 1958.
[23] K. Crammer, O. Dekel, J. Keshet, S. Shalev-Shwartz, and Y. Singer, "Online passive-aggressive algorithms," *J. Mach. Learn. Res.*, vol. 7, p. 551–585, dec 2006.
[24] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.
[25] H. Robbins and S. Monro, "A stochastic approximation method," *The annals of mathematical statistics*, pp. 400–407, 1951.