Very Fast Streaming Submodular Function Maximization

Sebastian Buschjäger[0000−0002−2780−3618](✉), Philipp-Jan Honysz[0000−1111−2222−3333], Lukas Pfahler[0000−1111−2222−3333], and Katharina Morik[0000−0003−1153−5986]

Artificial Intelligence Group, TU Dortmund, Germany
{`sebastian.buschjaeger, philipp.honysz, lukas.pfahler, katharina.morik`}`@tu-dortmund.de`

Abstract. Data summarization has become a valuable tool in understanding even terabytes of data. Due to their compelling theoretical properties, submodular functions have been the focus of summarization algorithms. Submodular function maximization is a well-studied problem with a variety of algorithms available. These algorithms usually offer worst-case guarantees to the expense of higher computation and memory requirements. However, many practical applications do not fall under this mathematical worst-case but are usually much more well-behaved. We propose a new submodular function maximization algorithm called ThreeSieves that ignores the worst-case and thus uses fewer resources. Our algorithm selects the most informative items from a data-stream on the fly and maintains a provable performance in most cases on a fixed memory budget. In an extensive evaluation, we compare our method against 6 state-of-the-art algorithms on 8 different datasets including data with and without concept drift. We show that our algorithm outperforms the current state-of-the-art in the majority of cases and, at the same time, uses fewer resources.

Keywords: Submodular Function Maximization · Streaming Data · Data Summarization.

1 Introduction

In recent years, submodular optimization has found its way into the toolbox of machine learning and data mining. Submodular functions reward adding a new element to a smaller set more than adding the same element to a larger set. This makes them ideal for solving data summarization tasks [22], active learning [28], user recommendation [1], and many other related tasks. In these tasks, the amount of data is often huge and generated in real-time. Consequently, a line of research studies streaming algorithms for maximizing a submodular function.

In this paper, we consider the problem of maximizing a submodular function over a data stream and focus on the task of data summarization. More formally, we consider the problem of selecting K representative elements from a ground set

V into a summary set $S \subseteq V$. To do so, we maximize a non-negative, monotone submodular set function $f\colon 2^V \to \mathbb{R}_+$ which assigns a utility score to each subset:

$$S^* = \underset{S \subseteq V, |S| = K}{\arg\max}\ f(S) \tag{1}$$

For the empty set, we assume zero utility $f(\emptyset) = 0$. We denote the maximum of f with $OPT = f(S^*)$. A set function can be associated with a marginal gain which represents the increase of $f(S)$ when adding an element $e \in V$ to S:

$$\Delta_f(e|S) = f(S \cup \{e\}) - f(S)$$

We call f submodular iff for all $A \subseteq B \subseteq V$ and $e \in V \setminus B$ it holds that

$$\Delta_f(e|A) \geq \Delta_f(e|B)$$

The function f is called monotone, iff for all $e \in V$ and for all $S \subseteq V$ it holds that $\Delta_f(e|S) \geq 0$.

The maximization of a submodular set function is NP-hard [11] and therefore, a natural approach is to find an approximate solution. Table 1 gives an overview of streaming algorithms which have been proposed for solving Eq. 1. To this date, the best performing online algorithms offer an $\mathcal{O}(\frac{1}{2} - \varepsilon)$ approximation ratio where ε also influences the resource consumption. Even moderate choices for ε quickly result in unmanageable resource consumption. Feldman et al. showed [12] that this approximation ratio is the best possible for streaming algorithms and that any algorithm with a better worst-case approximation guarantee essentially stores all the elements of the stream (up to a polynomial factor in K).

We ask whether we can design an algorithm that – despite the negative result from Feldman et al. – offers a *better* approximation ratio using *fewer* resources. Existing algorithms are designed for the mathematical *worst-case* and thereby have a worst-case approximation guarantee. We note, that this worse-cast is often a pathological case in their mathematical analysis whereas practical applications are usually much more well-behaved. Thus, we propose to *ignore* these pathological cases and derive an algorithm with a *better* approximation guarantee in *most* cases. Our proposed ThreeSieves algorithm estimates the probability of finding a more informative data item on the fly and only adds those items to the summary which are likely to not be 'out-valued' in the future. The resulting algorithm offers a *non-deterministic* approximation ratio of $(1 - \varepsilon)(1 - 1/\exp(1)) > \frac{1}{2} - \varepsilon$ in high probability $(1 - \alpha)^K$, where α is the desired user certainty. It performs $\mathcal{O}(1)$ function queries per item and requires $\mathcal{O}(K)$ memory. Note, that this does not contradict the upper bound of $\frac{1}{2} - \varepsilon$ since our algorithm offers a better approximation quality *in high probability*, but not deterministically for *all* cases. Our contributions are the following:

– The novel ThreeSieves algorithm has an approximation guarantee of $(1 - \varepsilon)(1 - 1/\exp(1))$ in high probability. The fixed memory budget is independent of ε storing at most K elements and the number of function queries is just one per element.

– For the first time we apply submodular function maximization algorithms to data containing concept drift. We show that our ThreeSieves algorithm offers competitive performance in this setting despite its weaker theoretical guarantee.
– We compare our algorithm against 6 state of the art algorithms on 8 datasets with and without concept drift. To the best of our knowledge, this is the first extensive evaluation of state-of-the-art submodular function maximization algorithms in a streaming setting. We show that ThreeSieves outperforms the current state-of-the-art in many cases while being up to 1000 times faster using a fraction of its memory.

The paper is organized as follows. Section 2 surveys related work, whereas in section 3 we present our main contribution, the ThreeSieves algorithm. Section 4 experimentally evaluates ThreeSieves. Section 5 concludes the paper.

Table 1: Algorithms for non-negative, monotone submodular maximization with cardinality constraint K. ThreeSieves offers the smallest memory consumption and the smallest number of queries per element in a streaming-setting.

Algorithm	**Approximation Ratio**	**Memory**	**Queries per Element**	**Stream**	**Ref.**
Greedy	$1 - 1/\exp(1)$	$\mathcal{O}(K)$	$\mathcal{O}(1)$	✗	[23]
StreamGreedy	$1/2 - \varepsilon$	$\mathcal{O}(K)$	$\mathcal{O}(K)$	✗	[13]
PreemptionStreaming	$1/4$	$\mathcal{O}(K)$	$\mathcal{O}(K)$	✓	[4]
IndependentSetImprovement	$1/4$	$\mathcal{O}(K)$	$\mathcal{O}(1)$	✓	[8]
Sieve-Streaming	$1/2 - \varepsilon$	$\mathcal{O}(K \log K/\varepsilon)$	$\mathcal{O}(\log K/\varepsilon)$	✓	[2]
Sieve-Streaming++	$1/2 - \varepsilon$	$\mathcal{O}(K/\varepsilon)$	$\mathcal{O}(\log K/\varepsilon)$	✓	[16]
Salsa	$1/2 - \varepsilon$	$\mathcal{O}(K \log K/\varepsilon)$	$\mathcal{O}(\log K/\varepsilon)$	(✓)	[24]
QuickStream	$1/(4c) - \varepsilon$	$\mathcal{O}(cK \log K \log(1/\varepsilon))$	$\mathcal{O}(\lceil 1/c \rceil + c)$	✓	[18]
ThreeSieves	$(1-\varepsilon)(1-1/\exp(1))$ with prob. $(1-\alpha)^K$	$\mathcal{O}(K)$	$\mathcal{O}(1)$	✓	this paper

2 Related Work

For a general introduction to submodular function maximization, we refer interested readers to [17] and for a more thorough introduction into the topic of streaming submodular function maximization to [9]. Most relevant to this publication are non-negative, monotone submodular streaming algorithms with cardinality constraints. There exist several algorithms for this problem setting which we survey here. The theoretical properties of each algorithm are summarized in Table 1. A detailed formal description including the pseudo-code of each algorithm is given in the appendix.
While not a streaming algorithm, the Greedy algorithm [23] forms the basis of many algorithms. It iterates K times over the entire dataset and greedily selects that element with the largest marginal gain $\Delta_f(e|S)$ in each iteration. It offers a $(1-(1/\exp(1))) \approx 63\%$ approximation and stores K elements. StreamGreedy

[13] is its adaption to streaming data. It replaces an element in the current summary if it improves the current solution by at-least ν. It offers an $\frac{1}{2} - \varepsilon$ approximation with $\mathcal{O}(K)$ memory, where ε depends on the submodular function and some user-specified parameters. The optimal approximation factor is only achieved if multiple passes over the data are allowed. Otherwise, the performance of StreamGreedy degrades arbitrarily with K (see Appendix of [2] for an example). We therefore consider StreamGreedy not to be a proper streaming algorithm.
Similar to StreamGreedy, PremptionStreaming [4] compares each marginal gain against a threshold $\nu(\mathcal{S})$. This time, the threshold dynamically changes depending on the current summary $\mathcal{S}$ which improves the overall performance. It uses constant memory and offers an approximation guarantee of 1/4. It was later shown that this algorithm is outperformed by SieveStreaming++ (see below) and was thus not further considered our experiments. Chakrabarti and Kale propose in [8] a streaming algorithm also with approximation guarantee of 1/4. Their algorithm stores the marginal gain of each element upon its arrival and uses this 'weight' to measure the importance of each item. We call this algorithm IndependentSetImprovement.
Norouzi-Fard et al. propose in [24] a meta-algorithm for submodular function maximization called Salsa which uses different algorithms for maximization as sub-procedures. The authors argue, that there are different types of data-streams and for each stream type, a different thresholding-rule is appropriate. The authors use this intuition to design a r-pass algorithm that iterates r times over the entire dataset and adapts the thresholds between each run. They show that their approach is a $(r/(r+1))^r - \varepsilon$ approximation algorithm. For a streaming setting, i.e. $r = 1$, this algorithm recovers the $1/2 - \varepsilon$ approximation bound. However note, that some of the thresholding-rules require additional information about the data-stream such as its length or density. Since this might be unknown in a real-world use-case this algorithm might not be applicable in all scenarios.
The first proper streaming algorithm with $1/2 - \varepsilon$ approximation guarantee was proposed by Badanidiyuru et al. in [2] and is called SieveStreaming. SieveStreaming tries to estimate the potential gain of a data item before observing it. Assuming one knows the maximum function value OPT beforehand and $|S| < K$, an element e is added to the summary S if the following holds:

$$\Delta_f(e|S) \geq \frac{OPT/2 - f(S)}{K - |S|} \tag{2}$$

Since OPT is unknown beforehand one has to estimate it before running the algorithm. Assuming one knows the maximum function value of a singleton set $m = max_{e \in V} f(\{e\})$ beforehand, then the optimal function value for a set with K items can be estimated by submodularity as $m \leq OPT \leq K \cdot m$. The authors propose to manage different summaries in parallel, each using one threshold from the set $O = \{(1+\varepsilon)^i \mid i \in \mathbb{Z}, m \leq (1+\varepsilon)^i \leq K \cdot m\}$, so that for at least one $v \in O$ it holds: $(1-\varepsilon)OPT \leq v \leq OPT$. In a sense, this approach sieves out elements with marginal gains below the given threshold - hence the authors name their approach SieveStreaming. Note, that this algorithm requires the knowledge of

$m = max_{e \in V} f(\{e\})$ before running the algorithm. The authors also present an algorithm to estimate m on the fly which does not alter the theoretical performance of SieveStreaming. Recently, Kazemi et al. proposed in [16] an extension of the SieveStreaming called SieveStreaming++. The authors point out, that the currently best performing sieve $S_v = \arg\max_v\{f(S_v)\}$ offers a better lower bound for the function value and they propose to use $[\max_v\{f(S_v)\}, K \cdot m]$ as the interval for sampling thresholds. This results in a more dynamic algorithm, in which sieves are removed once they are outperformed by other sieves and new sieves are introduced to make use of the better estimation of OPT. SieveStreaming++ does not improve the approximation guarantee of SieveStreaming, but only requires $\mathcal{O}(K/\varepsilon)$ memory instead of $\mathcal{O}(K \log K/\varepsilon)$. Last, Kuhnle proposed the QuickStream algorithm in [18] which works under the assumption that a single function evaluation is very expensive. QuickStream buffers up to c elements and only evaluates f every c elements. If the function value is increased by the c elements, thy all are added to the solution. Additionally, older examples are removed if there are more than K items in the solution. QuickStream performs well if the evaluation of f is very costly and if it is ideally independent from the size of the current solution S. This is unfortunately not the case in our experiments (see below). Moreover, QuickStream has a guarantee of $1/(4c) - \varepsilon$ that is outperformed by SieveStreaming(++) with similar resource consumption. Thus, we did not consider QuickStream in our experiments.

3 The Three Sieves Algorithm

We recognize, that SieveStreaming and its extensions offer a worst-case guarantee on their performance and indeed they can be consider optimal providing an approximation guarantee of $\frac{1}{2} - \varepsilon$ under polynomial memory constraints [12]. However, we also note that this worst case often includes pathological cases, whereas practical applications are usually much more well-behaved. One common practical assumption is, that the data is generated by the same source and thus follows the same distribution (e.g. in a given time frame). In this paper, we want to study these better behaving cases more carefully and present an algorithm which improves the approximation guarantee, while reducing memory and runtime costs in these cases. More formally, we will now assume that the items in the given sample (batch processing) or in the data stream (stream processing) are independent and identically distributed (iid). Note, that we do *not* assume any specific distribution. For batch processing this means, that all items in the data should come from the same (but unknown) distribution and that items should not influence each other. From a data-streams perspective this assumptions means, that the data source will produce items from the same distribution which does not change over time. Hence, we specifically *ignore* concept drift and assume that an appropriate concept drift detection mechanism is in place, so that summaries are e.g. re-selected periodically. We will study streams with drift in more detail in our experimental evaluation. We now use this assumption to derive an algorithm with $(1-\varepsilon)(1-1/\exp(1))$ approximation guarantee in high probability:

SieveStreaming and its extension, both, manage $\mathcal{O}(\log K/\varepsilon)$ sieves in parallel, which quickly becomes unmanageable even for moderate choices of K and ε. We note the following behavior of both algorithms: Many sieves in SieveStreaming have *too small a novelty-threshold* and quickly fill-up with uninteresting events. SieveStreaming++ exploits this insight by removing small thresholds early and thus by focusing on the most promising sieves in the stream. On the other hand, both algorithms manage sieves with *too large a novelty-threshold*, so that they never include any item. Thus, there are only a few thresholds that produce valuable summaries. We exploit this insight with the following approach: Instead of using many sieves with different thresholds we use only a single summary and carefully calibrate the threshold. To do so, we start with a large threshold that rejects most items, and then we gradually reduce this threshold until it accepts some - hopefully the most informative - items. The set $O = \{(1+\varepsilon)^i \mid i \in \mathbb{Z}, m \leq (1+\varepsilon)^i \leq K \cdot m\}$ offers a somewhat crude but sufficient approximation of OPT (c.f. [2]). We start with the largest threshold in O and decide for each item if we want to add it to the summary or not. If we do not add any of T items (which will be discussed later) to S we may lower the threshold to the next smallest value in O and repeat the process until S is full.

The key question now becomes: How to choose T appropriately? If T is too small, we will quickly lower the threshold and fill up the summary before any interesting item arrive that would have exceeded the original threshold. If T is too large, we may reject interesting items from the stream. Certainly, we cannot determine with absolute certainty when to lower a threshold without knowing the rest of the data stream or knowing the ground set entirely, but we can do so with high probability. More formally, we aim at estimating the probability $p(e|f,S,v)$ of finding an item e which exceeds the novelty threshold v for a given summary S and function f. Once p drops below a user-defined certainty margin τ

$$p(e|f,S,v) \leq \tau$$

we can safely lower the threshold. This probability must be estimated on the fly. Most of the time, we reject e so that S and $f(S)$ are unchanged and we keep estimating $p(e|f,S,v)$ based on the negative outcome. If, however, e exceeds the current novelty threshold we add it to S and $f(S)$ changes. In this case, we do not have any estimates for the new summary and must start the estimation of $p(e|f,S,v)$ from scratch. Thus, with a growing number of rejected items $p(e|f,S,v)$ tends to become close to 0 and the key question is how many observations do we need to determine – with sufficient evidence – that $p(e|f,S,v)$ will be 0.

The computation of confidence intervals for estimated probabilities is a well-known problem in statistics. For example, the confidence interval of binominal distributions can be approximated with normal distributions, Wilson score intervals, or Jeffreys interval. Unfortunately, these methods usually fail for probabilities near 0 [3]. However, there exists a more direct way of computing a confidence interval for heavily one-sided binominal distribution with probabilities near zero and iid data [15]. The probability of not adding one item in T trials is:

$$\alpha = (1 - p(e|f, S, v))^T \Leftrightarrow \ln(\alpha) = T \ln(1 - p(e|f, S, v))$$

A first order Taylor Approximation of $\ln(1 - p(e|f, S, v))$ reveals that $\ln(1 - p(e|f, S, v)) \approx -p(e|f, S, v)$ and thus $\ln(\alpha) \approx T(-p(e|f, S, v))$ leading to:

$$\frac{-\ln(\alpha)}{T} \approx p(e|f, S, v) \leq \tau \tag{3}$$

Therefore, the confidence interval of $p(e|f, S, v)$ after observing T events is $\left[0, \frac{-\ln(\alpha)}{T}\right]$. The 95% confidence interval of $p(e|f, S, v)$ is $\left[0, -\frac{\ln(0.05)}{T}\right]$ which is approximately $[0, 3/T]$ leading to the term “Rule of Three” for this estimate [15]. For example, if we did not add any of $T = 1000$ items to the summary, then the probability of adding an item to the summary in the future is below 0.003 given a $1 - \alpha = 0.95$ confidence interval. We can use the Rule of Three to quantify the certainty that there is a very low probability for finding a novel item in the data stream after observing T items. Note that we can either set α, τ and use Eq. 3 to compute the appropriate T value. Alternatively, we may directly specify T as a user parameter instead of α and τ, thereby effectively removing one hyperparameter. We call our algorithm ThreeSieves and it is depicted in Algorithm 1. Its theoretical properties are presented in Theorem 1.

```
**Input:** Stream $e_1, e_2, \ldots$, submodular function $f$ and parameters $K, T \in \mathbb{N}_{>0}$
**Output:** A set $S$ with at-most $K$ elements maximizing $f(S)$
$O \leftarrow \{(1+\varepsilon)^i \mid i \in \mathbb{Z}, m \leq (1+\varepsilon)^i \leq K \cdot m\}$
$v \leftarrow \max(O);\ O \leftarrow O \setminus \{\max(O)\};\ S \leftarrow \emptyset;\ t \leftarrow 0$
for next item e do
    if $\Delta_f(e|S) \geq \frac{v/2 - f(S)}{K - |S|}$ and $|S| < K$ then
        $S \leftarrow S \cup \{e\};\ t \leftarrow 0$
    else
        $t \leftarrow t + 1$
        if $t \geq T$ then
            $v \leftarrow \max(O);\ O \leftarrow O \setminus \{\max(O)\}; t \leftarrow 0$
        end
    end
end
return $S$
```

Algorithm 1: ThreeSieves algorithm.

Theorem 1. *ThreeSieves has the following properties:*

– *Let $K \in \mathbb{N}_{>0}$ be the maximum desired cardinality and let $1.0 - \alpha$ be the desired confidence interval. Given a fixed groundset V or an infinite data-stream in*

which each item is independent and identically distributed (iid) it outputs a set S such that $|S| \leq K$ and with probability $(1-\alpha)^K$ it holds for a non-negative, monotone submodular function $f\colon f(S) \geq (1-\varepsilon)(1-1/\exp(1))OPT$
- *It does 1 pass over the data (streaming-capable) and stores at most $\mathcal{O}(K)$ elements*

Proof. The detailed proof can be found in the appendix. The proof idea is as follows: The Greedy Algorithm selects that element with the largest marginal gain in each round. Suppose we know the marginal gains $v_i = \Delta(S|e_i)$ for each element $e_i \in S$ selected by Greedy in each round. Then we can simulate the Greedy algorithm by stacking K copies of the dataset consecutively and by comparing the gain $\Delta_f(e|S)$ of each element $e \in V$ against the respective marginal gain v_i. Let O be a set of estimated thresholds with $v_1^*, \dots, v_K^* \in O$. Let v_1^* denote the first threshold used by ThreeSieves before any item has been added to S. By the statistical test of ThreeSieves the $1-\alpha$ confidence interval for $P(v_1 \neq v_1^*)$ is given by $\frac{-\ln(\alpha)}{T}$. Differently coined, it holds with probability $1-\alpha$ that

$$P(v_1 \neq v_1^*) \leq \frac{-\ln(\alpha)}{T} \Leftrightarrow P(v_1 = v_1^*) > 1 - \frac{-\ln(\alpha)}{T}$$

Now we apply the confidence interval K times for each $P(v_j \neq v_j^*)$ individually. Then it holds with probability $(1-\alpha)^K$

$$P\left(v_1 = v_1^*, \dots, v_K = v_K^*\right) > \left(1 - \frac{-\ln(\alpha)}{T}\right)^K$$

By the construction of O it holds that $(1-\varepsilon)v_i^* \leq v_i \leq v_i^*$ (c.f. [2]). Let e_K be the element that is selected by ThreeSieves after $K-1$ items have already been selected. Let $S_0 = \emptyset$ and recall that by definition $f(\emptyset) = 0$, then it holds with probability $(1-\alpha)^K$:

$$\begin{aligned} f(S_K) &= f(\emptyset) + \sum_{i=1}^{K} \Delta(e_i|S_{K-1}) \geq \sum_{i=1}^{K} (1-\varepsilon)\, v_i^* \\ &= (1-\varepsilon)\, f_G(S_K) \geq (1-\varepsilon)\left(1 - 1/\exp(1)\right) OPT \end{aligned}$$

where f_G denotes the solution of the Greedy algorithm. □

Similar to SieveStreaming, ThreeSieves tries different thresholds until it finds one that fits best for the current summary S, the data V, and the function f. In contrast, however, ThreeSieves is optimized towards minimal memory consumption by maintaining one threshold and one summary at a time. If more memory is available, one may improve the performance of ThreeSieves by running multiple instances of ThreeSieves in parallel on different sets of thresholds. So far, we assumed that we know the maximum singleton value $m = max_{e \in V} f(\{e\})$ beforehand. If this value is unknown before running the algorithm we can estimate it on-the-fly without changing the theoretical guarantees of ThreeSieves.

As soon as a new item arrives with a new $m_{new} > m_{old}$ we remove the current summary and use the new upper bound $K \cdot m_{new}$ as the starting threshold. It is easy to see that this does not affect the theoretical performance of ThreeSieves: Assume that a new item arrives with a new maximum single value m_{new}. Then, all items in the current summary have a smaller singleton value $m_{old} < m_{new}$. The current summary has been selected based on the assumption that m_{old} was the largest possible value, which was invalidated as soon as m_{new} arrived. Thus, the probability estimate that the first item in the summary would be 'out-valued' later in the stream was wrong since we just observed that it is being out-valued. To re-validate the original assumption we delete the current summary entirely and re-start the summary selection.

4 Experimental Evaluation

In this section, we experimentally evaluate ThreeSieves and compare it against SieveStreaming(++), IndepdendentSetImprovement, Slasa, and Greedy. As an additional baseline we also consider a random selection of items via Reservoir Sampling [27]. We denote this algorithm as Random. We focus on two different application scenarios: In the first experiment, we have given a batch of data and are tasked to compute a comprehensive summary. In this setting, algorithms are allowed to perform multiple passes over the data to the expense of a longer runtime, e.g. as Greedy does. In the second experiment we shift our focus towards the summary selection on streams with concept drift. Here, each item is only seen once and the algorithms do not have any other information about the data-stream. All experiments have been run on an Intel Core i7-6700 CPU machine with 8 cores and 32 GB main memory running Ubuntu 16.04. The code for our experiments is available at `https://github.com/sbuschjaeger/SubmodularStreamingMaximization`

4.1 Batch experiments

In the batch experiments each algorithm is tasked to select a summary with exactly K elements. Since most algorithms can reject items they may select a summary with less than K elements. To ensure a summary of size K, we re-iterate over the entire data-set as often as required until K elements have been selected, but at most K times. We compare the relative maximization performance of all algorithms to the solution of Greedy. For example, a relative performance of 100% means that the algorithm achieved the same performance as Greedy did, whereas a relative performance of 50% means that the algorithm only achieved half the function value of Greedy. We also measure the total runtime and memory consumption of each algorithm. The runtime measurements include all re-runs, so that many re-runs over the data-set result in larger runtimes. To make our comparison implementation independent, we report the *algorithmic* memory consumption in terms of the total number of items stored by each algorithm. For example, the Forestcover dataset contains observations with $d = 10$ features which can be each be represented using a 4 byte `float` variable. Thus, an algorithm that stores 4096 of these observations uses $4096 \cdot 4 \cdot d/1024 = 160$ KB of memory in this specific instance. We evaluate four key questions: First, is ThreeSieves competitive against the other algorithms or will the probabilistic guarantee hurt its practical performance? Second, if ThreeSieves is competitive, how does it related to a Random selection of summaries? Third, how large is the resource consumption of ThreeSieves in comparison? Fourth, how does ThreeSieves behave for different T and different ε?

In total, we evaluate 3895 hyperparameter configurations on the datasets shown in the top group in Table 2. We extract summaries of varying sizes $K \in \{5, 10, \ldots, 100\}$ maximizing the log-determinant $f(S) = \frac{1}{2}\log\det(\mathcal{I} + a\Sigma_S)$. Here, $\Sigma_S = [k(e_i, e_j)]_{ij}$ is a kernel matrix containing all similarity pairs of all points in S, $a \in \mathbb{R}_+$ is a scaling parameter and $\mathcal{I}$ is the identity matrix. In [26], this function is shown to be submodular. Its function value does not depend on V, but only on the summary S, which makes it an ideal candidate for summarizing data in a streaming setting. In [5], it is proven that $m = max_{e \in V} f(\{e\}) = 1 + aK$ and that $OPT \leq K \log(1 + a)$ for kernels with $k(\cdot, \cdot) \leq 1$. This property can be enforced for every positive definite kernel with normalization [14]. In our experiments we set $a = 1$ and use the RBF kernel $k(e_i, e_j) = \exp\left(-\frac{1}{2l^2} \cdot ||e_i - e_j||_2^2\right)$ with $l = \frac{1}{2\sqrt{d}}$ where d is the dimensionality of the data. We vary $\varepsilon \in \{0.001, 0.005, 0.01, 0.05, 0.1\}$ and $T \in \{50, 250, 500, 1000, 2500, 5000\}$.

Table 2: Data sets used for the experiments.

Name	Size	Dim	Reference
ForestCover	286,048	10	[10]
Creditfraud	284,807	29	[21]
FACT Highlevel	200,000	16	[6]
FACT Lowlevel	200,000	256	[6]
KDDCup99	60,632	41	[7]
stream51	150,736	2048	[25]
abc	1,186,018	300	[20]
examiner	3,089,781	300	[19]

We present two different sets of plots, one for varying K and one for varying ε. Additional plots with more parameter variations are given in the appendix. Figure 1 depicts the relative performance, the runtime and the memory consumption over different K for a fixed $\varepsilon = 0.001$. For presentational purposes, we selected $T = 500, 1000, 2500, 5000$ for ThreeSieves. In all experiments, we find that ThreeSieves with $T = 5000$ and Salsa generally perform best with a very close performance to Greedy for $K \geq 20$. For smaller summaries with $K < 20$

all algorithms seem to underperform, with Salsa and SieveStreaming performing best. Using $T \leq 1000$ for ThreeSieves seems to decrease the performance on some datasets, which can be explained by the probabilistic nature of the algorithm. We also observe a relative performance above 100 where ThreeSieves performed *better* than Greedy on Creditfraud and Fact Highlevel. Note, that only ThreeSieves showed this behavior, whereas the other algorithms never exceeded Greedy. Expectantly, Random selection shows the weakest performance. SieveStreaming and SieveStreaming++ show identical behavior. Looking at the runtime, please, note the logarithmic scale. Here, we see that ThreeSieves and Random are by far the fastest methods. Using $T = 1000$ offers some performance benefit, but is hardly justified by the decrease in maximization performance, whereas $T = 5000$ is only marginally slower but offers a much better maximization performance. SieveStreaming and SieveStreaming++ have very similar runtime, but are magnitudes slower than Random and ThreeSieves. Last, Salsa is the slowest method. Regarding the memory consumption, please, note again the logarithmic scale. Here, all versions of ThreeSieves use the fewest resources as our algorithm only stores a single summary in all configurations. These curves are identical with Random and IndependentSetImprovement so that only four instead of 7 curves are to be seen. SieveStreaming and its siblings use roughly two magnitudes more memory since they keep track of multiple sieves in parallel.

Now we look at the behavior of the algorithms for different approximation ratios. Again we refer readers to the additional plots with more parameter variations in the appendix. Figure 2 depicts the relative performance, the runtime, and the memory consumption over different ε for a fixed $K = 50$. We again selected $T = 500, 1000, 2500, 5000$ for ThreeSieves. Note, that for Random, Greedy and IndependentSetImprovement there is now a constant curve since their performance does not vary with ε. Looking at the relative performance we see a slightly different picture than before: For small $\varepsilon \leq 0.05$ and larger T we see that ThreeSieves and Salsa again perform best in all cases. For larger $\varepsilon > 0.05$ the performance of the non-probabilistic algorithms remain relatively stable, but ThreeSieves performance starts to deteriorate. Again we note, that SieveStreaming and SieveStreaming++ show identical behavior. Looking at the runtime and memory consumption we see a similar picture as before: ThreeSieves is by far the fastest method using the fewest resources followed by SieveStreaming(++) and Salsa.

We conclude: In summary, ThreeSieves works best for small ε and large T. The probabilistic nature of the algorithm does not decrease its maximization performance but actually helps it in some cases. In contrast to the other algorithms, the resource consumption and overall runtime of ThreeSieves does not suffer from decreasing ε or increasing T. Additional experiments and comparisons can be found in the appendix. They all depict the same general trend discussed here. In particular, when comparing SieveStreaming(++) and Salsa with ThreeSieves under similar maximization performance (e.g using $\varepsilon = 0.1$ for SieveStreaming(++) and Salsa, and using $\varepsilon = 0.001$ for ThreeSieves) we still find that ThreeSieves uses magnitudes less resources. Its overall performance is

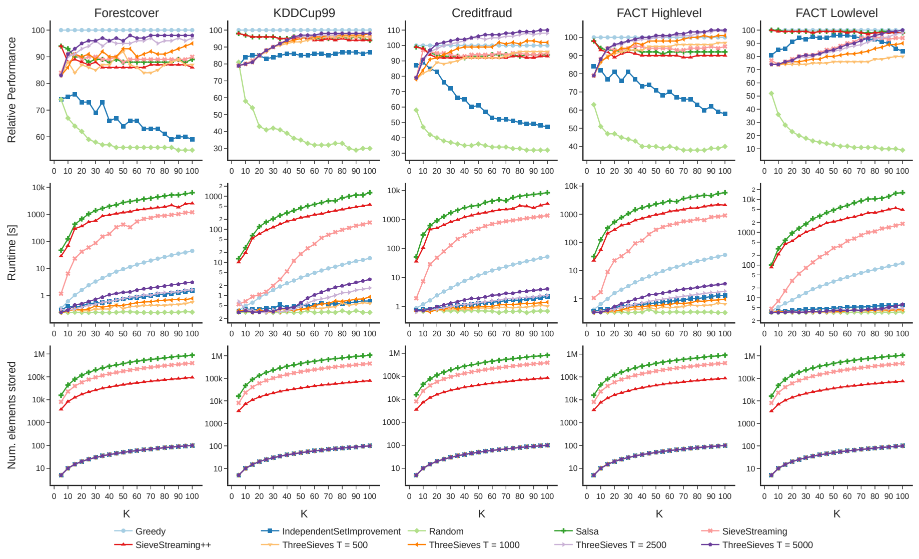

Fig. 1: Comparison between SieveStreaming,SieveStreaming++,Salsa, and ThreeSieves for different K values and fixed $\varepsilon = 0.001$. The first row shows the relative performance to Greedy (larger is better), the second row shows the total runtime in seconds (logarithmic scale, smaller is better) and the third row shows the maximum memory consumption (logarithmic scale, smaller is better). Each column represents one dataset.

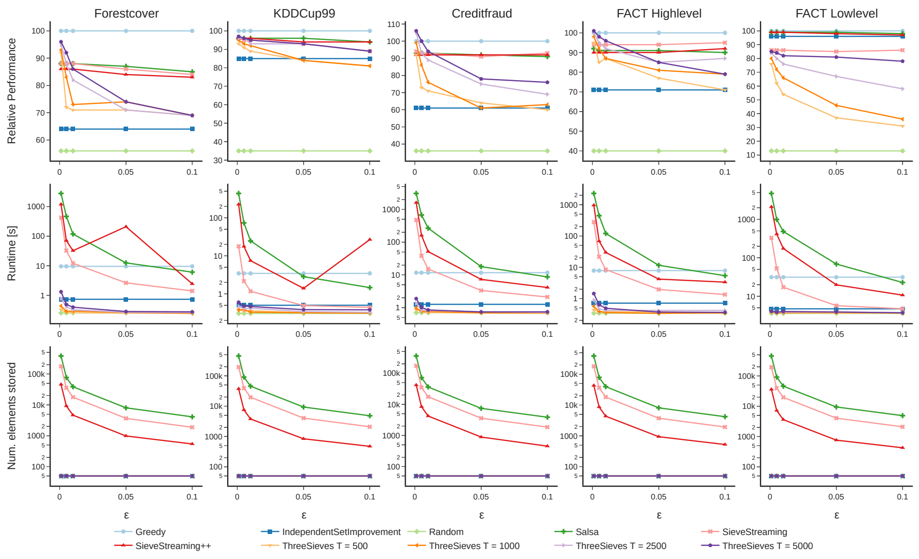

Fig. 2: Comparison between IndependentSetImprovement, SieveStreaming,SieveStreaming++,Salsa, Random, and ThreeSieves for different ε values and fixed $K = 50$. The first row shows the relative performance to Greedy (larger is better), the second row shows the total runtime in seconds (logarithmic scale, smaller is better) and the third row shows the maximum memory consumption (logarithmic scale, smaller is better). Each column represents one dataset.

better or comparable to the other algorithms while being much more memory efficient and overall faster.

4.2 Streaming experiments

Now we want to compare the algorithms in a true streaming setting including concept drift. Here, we present each item only once and must decide immediately if it should be added to the summary or not. Since Salsa requires additional information about the stream we excluded it for these experiments. We use two real-world data-sets and one artificial data-set depicted in the bottom group in Table 2: The stream51 dataset [25] contains image frames from a sequence of videos, where each video shows an object of one of 51 classes. Subsequent frames in the video stream are highly dependent. Over the duration of the stream, new classes are introduced. The dataset is constructed such that online streaming classification methods suffer from 'Catastrophic forgetting' [25]. We utilize a pretrained InceptionV3 convolutional neural network that computes 2048-dimensional embeddings of the images. The abc dataset contain news headlines from the Australian news source 'ABC' gathered over 17 years (2003 - 2019) and the examiner dataset contains news headlines from the Australian news source 'The Examiner' gathered over 6 years (2010 - 2015). Due to this long time-span we assume that both datasets contain a natural concept drift occurring due to different topics in the news. We use pretrained Glove embeddings to extract 300-dimensional, real-valued feature vectors for all headlines and present them in order of appearance starting with the oldest headline. We ask the following questions: First, will the iid assumption of ThreeSieves hurt its practical performance on data-streams with concept drift? Second, how will the other algorithms with a worst-case guarantee perform in this situation?

In total, we evaluate 3780 hyperparameters on these three datasets. Again, we extract summaries of varying sizes $K \in \{5, 10, \dots, 100\}$ maximizing the log-determinant with $a = 1$. We use the RBF kernel with $l = \frac{1}{\sqrt{d}}$ where d is the dimensionality of the data. We vary $\varepsilon \in \{0.01, 0.1\}$ and $T \in \{500, 1000, 2500, 5000\}$. Again, we report the relative performance of the algorithms compared to Greedy (executed in a batch-fashion). Figure 3 shows the relative performance of the streaming algorithms for different K with fixed $\varepsilon = 0.1$ (first row) and fixed $\varepsilon = 0.01$ (second row) on the three datasets. On the stream51 dataset we see very chaotic behavior for $\varepsilon = 0.1$. Here, SieveStreaming++ generally seems to be best with a performance around $90 - 95\%$. ThreeSieves's performance suffers for smaller $T \leq 1000$ not exceeding 85%. For other configurations with larger T ThreeSieves has a comparable performance to SieveStreaming and IndependentSetImprovement all achieving $85 - 92\%$. For $\varepsilon = 0.01$ the behavior of the algorithms stabilizes. Here we find that ThreeSieves with $T = 5000$ shows a similar and sometimes even better performance compared to SieveStreaming(++) beyond 95%. An interesting case occurs for $T = 5000$ and $K = 100$ in which the function value suddenly drops. Here, ThreeSieves rejected more items than were available in the stream and thus returned a summary with less than $K = 100$ items. Somewhere in the middle, we find IndependentSetImprovement

reaching 90% performance in the best case. Expectantly, Random selection is the worst in all cases. In general, we find a similar behavior on the other two datasets. For $\varepsilon = 0.1$, SieveStreaming(++) seem to be the best option followed by ThreeSieves with larger T or IndependentSetImprovement. For larger K, IndependentSetImprovement is not as good as ThreeSieves and its performance approaches Random quite rapidly. For $\varepsilon = 0.01$ the same general behavior can be observed. SieveStreaming(++) again holds well under concept drift followed by ThreeSieves with larger T followed by IndependentSetImprovement and Random. We conjecture that ThreeSieves's performance could be further improved for larger T and that there seems to be a dependence between T and the type of concept drift occurring in the data.
We conclude: ThreeSieves holds surprisingly well under concept drift, especially for larger T. In many cases its maximization performance is comparable with SieveStreaming(++) while being more resource efficient. For smaller T the performance of ThreeSieves clearly suffers, but remains well over the performance of Random or IndependentSetImprovement. For larger T and smaller ε, ThreeSieves becomes more competitive to SieveStreaming(++) while using fewer resources. We conclude that ThreeSieves is also applicable for streaming data with concept drift even though its theoretical guarantee does not explicitly hold in this context.

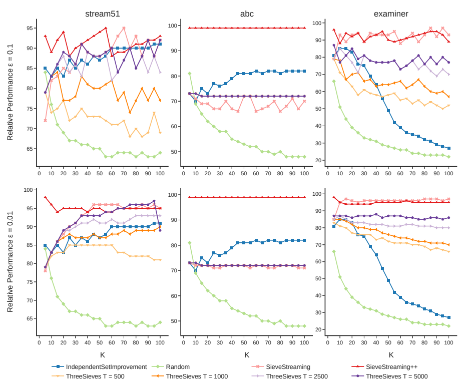

Fig. 3: Comparison between IndependentSetImprovement, SieveStreaming, SieveStreaming++, Random and ThreeSieves for different K values and fixed $\varepsilon = 0.1$ (first row) and $\varepsilon = 0.01$ (second row). Each column represents one dataset.

5 Conclusion

Data summarization is an emerging topic for understanding and analyzing large amounts of data. In this paper, we studied the problem of on-the-fly data summarization where one must decide immediately whether to add an item to the summary, or not. While some algorithms for this problem already exist, we recognize that these are optimized towards the worst-case and thereby require more resources. We argue that practical applications are usually much more well-behaved than the commonly analyzed worst-cases. We proposed the ThreeSieves algorithm for non-negative monotone submodular streaming function maximization and showed, that – under moderate assumptions – it has an approximation ratio of $(1-\varepsilon)(1-1/\exp(1))$ in high probability. It runs on a fixed memory budget and performs a constant number of function evaluations per item. We compared ThreeSieves against 6 state of the art algorithms on 8 different datasets with and without concept drift. For data without concept drift, ThreeSieves outperforms the other algorithms in terms of maximization performance while using two magnitudes less memory and being up to 1000 times faster. On datasets with concept drift, ThreeSieves outperforms the other algorithms in some cases and offers similar performance in the other cases while being much more resource efficient. This allows for applications, where based on the summary, some action has to be performed. Hence, the novel ThreeSieves algorithm opens up opportunities beyond the human inspection of data summaries, which we want to explore in the future.

Acknowledgements

Part of the work on this paper has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 "Providing Information by Resource-Constrained Analysis", DFG project number 124020371, SFB project A1, `http://sfb876.tu-dortmund.de`. Part of the work on this research has been funded by the Federal Ministry of Education and Research of Germany as part of the competence center for machine learning ML2R (01—18038A), `https://www.ml2r.de/`.

References

1. Ashkan, A., Kveton, B., Berkovsky, S., Wen, Z.: Optimal greedy diversity for recommendation. In: IJCAI. vol. 15, pp. 1742–1748 (2015)
2. Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., Krause, A.: Streaming submodular maximization: Massive data summarization on the fly. In: ACM SIGKDD (2014)
3. Brown, L.D., Cai, T.T., DasGupta, A.: Interval estimation for a binomial proportion. Statistical science pp. 101–117 (2001)
4. Buchbinder, N., Feldman, M., Schwartz, R.: Online submodular maximization with preemption. ACM Trans. Algorithms **15**(3) (Jun 2019)
5. Buschjäger, S., Morik, K., Schmidt, M.: Summary extraction on data streams in embedded systems. In: ECML Conference Workshop IoT Large Scale Learning from Data Streams (2017)
6. Buschjäger, S., Pfahler, L., Buss, J., Morik, K.: On-site gamma-hadron separation with deep learning on fpgas (accepted). In: Machine Learning and Knowledge Discovery in Databases (ECML PKDD). Springer International Publishing (2020)

7. Campos, G.O., Zimek, A., Sander, J., Campello, R.J., Micenková, B., Schubert, E., Assent, I., Houle, M.E.: On the evaluation of unsupervised outlier detection: measures, datasets, and an empirical study. DAMI **30**(4), 891–927 (2016)
8. Chakrabarti, A., Kale, S.: Submodular maximization meets streaming: Matchings, matroids, and more. In: Integer Programming and Combinatorial Optimization. pp. 210–221 (2014)
9. Chekuri, C., Gupta, S., Quanrud, K.: Streaming algorithms for submodular function maximization. In: International Colloquium on Automata, Languages, and Programming. pp. 318–330. Springer (2015)
10. Dal Pozzolo, A., Caelen, O., Johnson, R.A., Bontempi, G.: Calibrating probability with undersampling for unbalanced classification. In: 2015 IEEE SSCI. `https://www.kaggle.com/mlg-ulb/creditcardfraud`
11. Feige, U.: A threshold of ln n for approximating set cover. J. ACM **45**(4), 634–652
12. Feldman, M., Norouzi-Fard, A., Svensson, O., Zenklusen, R.: The one-way communication complexity of submodular maximization with applications to streaming and robustness. In: 52nd Annual ACM SIGACT STOC. pp. 1363–1374 (2020)
13. Gomes, R., Krause, A.: Budgeted nonparametric learning from data streams. In: ICML. vol. 1, p. 3 (2010)
14. Graf, A.B., Borer, S.: Normalization in support vector machines. In: DAGM Symposium of Pattern Recognition (2001)
15. Jovanovic, B.D., Levy, P.S.: A look at the rule of three. The American Statistician **51**(2), 137–139 (1997)
16. Kazemi, E., Mitrovic, M., Zadimoghaddam, M., Lattanzi, S., Karbasi, A.: Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. In: ICML. pp. 3311–3320 (2019)
17. Krause, A., Golovin, D.: Submodular function maximization. (2014)
18. Kuhnle, A.: Quick streaming algorithms for maximization of monotone submodular functions in linear time (2021)
19. Kulkarni, R.: The examiner - spam clickbait catalog – 6 years of crowd sourced journalism (2017), `https://www.kaggle.com/therohk/examine-the-examiner`
20. Kulkarni, R.: A million news headlines – news headlines published over a period of 17 years (2017), `https://www.kaggle.com/therohk/million-headlines`
21. Liu, F.T., Ting, K.M., Zhou, Z.H.: Isolation forest. In: 2008 Eighth IEEE International Conference on Data Mining. pp. 413–422. IEEE (2008), `http://odds.cs.stonybrook.edu/forestcovercovertype-dataset/`
22. Mirzasoleiman, B., Badanidiyuru, A., Karbasi, A.: Fast constrained submodular maximization: Personalized data summarization. In: ICML. pp. 1358–1367 (2016)
23. Nemhauser, G., et al.: An analysis of approximations for maximizing submodular set functions—i. Mathematical Programming (1978)
24. Norouzi-Fard, A., Tarnawski, J., Mitrovic, S., Zandieh, A., Mousavifar, A., Svensson, O.: Beyond 1/2-approximation for submodular maximization on massive data streams. In: ICML. pp. 3829–3838 (10–15 Jul 2018)
25. Roady, R., Hayes, T.L., Vaidya, H., Kanan, C.: Stream-51: Streaming classification and novelty detection from videos. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops (June 2020)
26. Seeger, M.: Greedy forward selection in the informative vector machine. Tech. rep., Technical report, University of California at Berkeley (2004)
27. Vitter, J.S.: Random sampling with a reservoir. ACM Transactions on Mathematical Software (TOMS) **11**(1), 37–57 (1985)
28. Wei, K., Iyer, R., Bilmes, J.: Submodularity in data subset selection and active learning. In: International Conference on Machine Learning. pp. 1954–1963 (2015)