Summary Extraction on Data Streams in Embedded Systems

Sebastian Buschjäger, Katharina Morik, and Maik Schmidt

TU Dortmund University
Computer Science VIII: Artificial Intelligence Unit
{sebastian.buschjaeger, katharina.morik}@tu-dortmund.de
http://www-ai.cs.uni-dortmund.de/

Abstract. More and more data is created by humans and cyber-physical systems having sensing, acting and networking capabilities. Together, these systems form the Internet of Things (IoT). The realtime analysis of its data may provide us with valuable insights about the complex inner processes of the IoT. Moreover, these insights offer new opportunities ranging from sensor monitoring to actor control. The volume and velocity of the data at the distributed nodes challenge human as well as machine monitoring of the IoT. Broadcasting all measurements to a central node might exceed the network capacity as well as the resources at the central node or the human attention span. Hence, data should be reduced already at the local nodes such that the submitted information can be used for efficient monitoring.
There are several methods that aim at data summarization ranging from clustering, aggregation to compression. Where most of the approaches transform the representation, we want to select unchanged data items from the data stream, already while they are generated by the cyber-physical system and at the cyber-physical system. The observations are selected independent of their frequencies. They are meant to be efficiently transmitted. The ideal case is that no important measurement is missing in the selection and that no redundant items are transmitted. The data summary is easily interpreted and is available in realtime. We focus on submodular function maximization due to its strong theoretical background. We investigate its use for data summarization and enhance the Sieve-Streaming algorithm for data summarization on data streams such that it delivers smaller sets with high recall.

Keywords: Data summarization, Embedded systems, Streaming Data

1 Introduction

With increasing processing capabilities, more and more data is gathered every day at virtually any place on earth. Most of this data is produced by small embedded electronics with sensing, acting and networking capabilities [11].

To capitalize on the vast amount of data produced by these IoT devices, machine learning has proven a valuable tool, but is usually limited to large server systems due to resource constraints. In order to speed up the analysis for monitoring, we wish to move parts of the analysis to the measuring device by the means of data summarization. For numerical data streams, moving averages have been used. For categorial values, data summarization describes the extraction of representative items from a data set and has been used for texts [7,8], speech [14], and images [13]. Informally, a summary should contain a few, but informative items, where each item reflects one hidden state or class of the underlying system. Given such a summary, we can use it to

- perform certain actions, once a new item is added to the summary
- compare summaries of different devices and points in time
- provide a quick overview for human experts

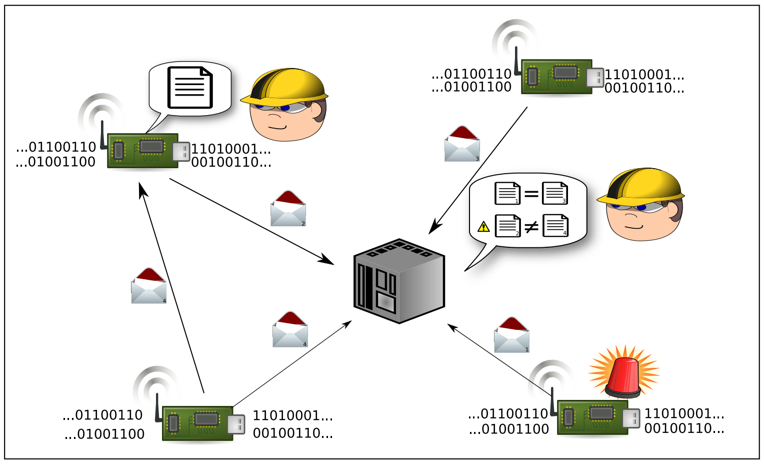


Fig. 1: Use cases of data summarization in IoT streaming settings. Each device gathers a data summary on its own and communicates it to central server or to other IoT devices. Human experts can examine summaries and perform actions if needed.

Data summarization has been viewed as a submodular function maximization problem, which offers an appealing theoretical framework with strong quality bounds. In this paper, we focus on data summarization on data streams by the means of submodular function maximization, namely the Sieve-Streaming algorithm [1].

The main question when computing a data summary is, what items are "novel" enough to be included into the summary and which are too similar to already seen items. If this novelty threshold is chosen too small, the summary will contain redundant items, whereas if the novelty is expected to be too high, we will not add any item to the summary. Sieve-Streaming deals with this challenge by managing multiple summaries in parallel, each with its own novelty threshold.

We will exploit the ideas of Sieve-Streaming for monitoring of distributed measuring devices. Hence, we first investigate, whether the Sieve-Streaming is appropriate for our setting. Additionally, we extend the algorithm for the use in embedded systems:

– We reduce the number of summaries by tighter bounds of the utility threshold used by Sieve-Streaming without sacrificing its theoretical guarantees.
– By dynamic thresholding, we further improve the quality of Sieve-Streaming without increasing its computational costs.
– We evaluate the enhancements empirically.

The paper is organized as the following. The next section will give a brief overview of the Sieve-Streaming algorithm. Section 3 shows our enhancements of the Sieve-Streaming algorithm in detail. Section 4 presents experiments on three different data sets to evaluate data summarization in general and our enhancements of Sieve-Streaming in detail. We conclude the paper in section 5.

2 Sieve-Streaming

In this section, we shortly present the Sieve-Streaming algorithm and introduce the notation used in this paper. A more detailed overview of submodular function maximization can be found in [4]. Submodular function maximization considers the problem of selecting a set $S \subseteq V$ with fixed size $|S| = K$ from a ground set V, such that a set function $f \colon 2^V \to \mathbb{R}$ is maximized, which assigns a utility score to each possible subset $S \subseteq V$. For set functions, the marginal gain is given by the increase in $f(S)$ when adding an element $e \in V$ to S:

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

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

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

Submodular functions incorporate a notion of diminishing returns: Adding a new element to a smaller set will increase the utility value more than adding the same element to a larger set. This property intuitively captures one aspect of summarization, in which we seek small but expressive summaries and thus adding elements to a summary should be done with care.

Now, let us apply monotone submodular function maximization in a streaming setting, so that data items e arrive one at a time and we have to decide immediately, whether we want to add e to S, or not. A number of different approaches for streaming settings have been proposed in literature with different constraints and assumptions about the data (see [9] for a very recent overview). The approach of Badanidiyuru and colleagues fits our settings best [1].

Submodularity and monotony allows us to greedily increase the utility value by simply adding new items to the summary. Since the maximum summary size is restricted to K, this approach simply picks the first K elements without considering other items in the stream. Thus, we should add items to the summary only, if they offer a reasonable increase in the summary's utility value, i.e. add enough novelty.

Let $OPT = max_{S \subseteq V, |S|=K} f(S)$ denote the maximum possible function value. Assume we know OPT beforehand and we have already selected some points in S. In this situation we could expect the next item e to add at least $\frac{OPT - f(S)}{K - |S|}$ to the utility so that we can reach OPT with the remaining $K - |S|$ elements. In a sense, this approach sieves out elements with marginal gains below the given threshold.
However, this approach does not work, if the optimal summary contains many elements with a gain just below the threshold and one element with large marginal gain. In this case, we could miss the items with smaller marginal gain waiting for an item with large gain. To counter this behavior, the authors of [1] propose to lower the threshold by $\frac{1}{2}$, adding e to S if the following holds:

$$\Delta_f(e|S) \geq \frac{OPT/2 - f(S)}{K - |S|} \tag{1}$$

Knowing the optimal utility value OPT is part of the problem we would like to solve. Usually, this value is unknown beforehand and needs to be estimated. In order to guarantee a good estimate we can generate multiple estimates for OPT before running the algorithm and manage multiple sieves each with their own threshold values in parallel.
We can narrow down the value of OPT by using the properties of submodularity. To do so, let $m = max_{e \in V} f(\{e\})$ denote the maximum value of a singleton item. Given e occurs at least once in the data stream, the worst optimal summary would contain e at least once. If e however occurs at least K times in the data stream, then the best summary would contain e K-times, so that:

$$m \leq OPT \leq K \cdot m \tag{2}$$

Therefore, knowing the maximum singleton value $f(\{e\})$ already gives a rough estimate of the range of values we can expect from f. This knowledge can now be used to sample different threshold values from the interval $[m, Km]$ such that one of these thresholds will be close to OPT. More formally, we 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: $(1-\varepsilon)OPT \leq v \leq OPT$. Algorithm 1 summaries the proposed method. It can be shown, that algorithm 1 extracts a summary with:

$$f(S) \geq (\frac{1}{2} - \varepsilon)OPT \tag{3}$$

2.1 Utility function

Given a summary $S = \{e_1, \ldots, e_K\}$ we can express the similarity between elements e_i and e_j using a kernel function $k(e_i, e_j)$, which gives rise to the kernel matrix $\Sigma = [k(e_i, e_j)]_{ij}$ containing all similarity pairs. This matrix has the largest

Algorithm 1 Sieve-Streaming with known singleton maximum value m.

```
 1: $O = \{(1+\varepsilon)^i \mid i \in \mathbb{Z}, m \leq (1+\varepsilon)^i \leq k \cdot m\}$
 2: for $v \in O$ do
 3:     $S_v = \emptyset$
 4: end for
 5: for next item $e$ do
 6:     for $v \in O$ do
 7:         if $\Delta_f(e|S_v) \geq \frac{v/2 - f(s)}{K-|S|}$ and $|S_v| < K$ then
 8:             $S_v = S_v \cup \{e\}$
 9:         end if
10:     end for
11: end for
```

values on the diagonal as items are the most similar to themselves, whereas values on the off-diagonal indicate the similarity between distinct elements and thus are usually smaller. Intuitively, we seek an expressive summary, so that Σ contains many values near 0 on its off-diagonal. This intuition is formally captured by the Informative Vector Machine [5] proposing the function:

$$f(S) = \frac{1}{2} logdet(I + \sigma \Sigma)$$

where I indicates the $K \times K$ identity matrix and $\sigma > 0$ denotes some scaling parameter. In [10] it was shown, that this function is monotone submodular.

3 Embedded Summary Extraction

In the previous section, we presented the Sieve-Streaming algorithm. Now, we will introduce the changes that make it better suited for summarization in embedded systems. First we show how to reduce the number of sieves without sacrificing performance guarantees. Second, we introduce dynamic thresholding which further increases the utility value. Last, we consider the trade-off between memory consumption and runtime for an efficient implementation.

Reduce the number of sieves: Sieve-Streaming needs to keep track of multiple sieves in parallel with corresponding thresholds chosen from the interval $[m, Km]$. By taking the special structure of f into account, we can reduce the size of this interval and thus the overall memory requirements. To do so, we assume that the kernel function $k(\cdot,\cdot)$ and its corresponding kernel matrix are positive definite.
The upper bound Km of the threshold v is sharp and thus cannot be improved. Intuitively, a summary reaches its maximum function value when all entries on the off-diagonal are 0. In this case $logdet(I + \sigma^2 \Sigma)$ equals $log((1+\sigma)^K) = Klog(1+\sigma) = K \cdot m$, which is exactly what we derived using submodularity.
The lower bound m, however, can be improved substantially: A summary reaches

its minimum utility when all entries on the off-diagonal equal the largest possible kernel value C, that is, we picked the same element K times. W.l.o.g we assume that $C = 1$ in the following, e.g. we use the RBF kernel $k(x, y) = exp(-\frac{1}{2}||x - y||^2)$. Note, that it is possible to scale every positive definite kernel to the interval $[0, 1]$ without damaging its positive definite property (cf. [3]). By using Sylvester's determinant theorem, we see that $logdet(I + \sigma\Sigma) = log(I+\sigma\mathbf{1}^T\mathbf{1}) = log(1+\sigma K) > m = log(1+\sigma)$ where $\mathbf{1}$ denotes the vector, where all entries are 1. In order to show that this intuition really creates a lower bound, we use the characteristic polynomial $det(\lambda I - (I + \sigma\Sigma)) = \sum_{i=1}^{K}(-1)^k\lambda^{K-i}a_i$. The determinant can be computed in terms of characteristic coefficients a_i, that is $det(I + \sigma\Sigma) = \sum_{i=0}^{K} a_i = 1 + det(\sigma\Sigma) + tr(\sigma\Sigma) + \sum_{i=2}^{K-1} a_i$. Since $I + \sigma\Sigma$ is positive definite, each coefficient a_i can be expressed as the sum of eigenvalues of $I + \sigma\Sigma$, which are also all positive [2]. Since $\sigma\Sigma$ is also positive definite, we see that $det(\sigma\Sigma) \geq 0$. This leads then to

$$det(I + \sigma\Sigma) = 1 + det(\sigma\Sigma) + tr(\sigma\Sigma) + \sum_{i=2}^{K-1} a_i \geq 1 + tr(\sigma\Sigma) = 1 + \sigma K$$

and thus: $logdet(I + \sigma\Sigma) \geq log(1 + \sigma K)$.

Dynamic Thresholding: Sieve-Streaming creates a fixed number of sieves $|\mathcal{O}|$ in the beginning of the algorithm. Due to the thresholding approach, sieves with smaller thresholds will accept more items and thus fill-up more quickly, whereas sieves with large thresholds are more picky. Once a summary is full the corresponding sieve is not used anymore. This allows us to create another sieve in its place with a different threshold while keeping the overall number of sieves constant.

In turn, we can use this new threshold to refine the estimate of OPT. Let v_c denote the largest threshold corresponding to a full summary and let v_o denote the smallest threshold corresponding to a non-full summary. Note that by construction $v_o > v_c$ and that - given the current summaries - OPT seems to be in the interval $[v_c, v_o]$. Therefore, we can refine the estimate of OPT if we create a new sieve in $[v_c, v_o]$.

With this approach "older" sieves with larger threshold may contain more items than "newer" sieves. In turn, one of these "older" sieves may become full at one point, whereas the newly created sieves with smaller threshold are still accepting elements. Since the thresholds for these sieves are smaller than that the sieve that just became full, we know that these sieves will not offer any better utility values. Therefore, we can close these sieves once a sieve with larger threshold closes.

In practice one rather unintuitive effect may happen, in which all summaries of sieves might be full at some point in time. In pure Sieve-Streaming, sieves are created in the interval $[m, Km]$, where the upper bound Km represents the largest utility possible. This bound is independent from the data and therefore we expect that at least one sieve will be open all the time aiming for an extreme case with utility Km.

Recall however, that only the half of each threshold (see eq. 1) is used when the marginal gain of a new item is evaluated. Thus, Sieve-Streaming effectively uses $\frac{1}{2}Km$ as upper bound. This is enough to ensure a $(\frac{1}{2} - \varepsilon)$ approximation, but might be sub-optimal in some cases. To counter this, we gradually increase the interval of thresholds to $2Km$, once we notice that all sieves are closed.

Implementation: In a naive approach, each sieve needs to store K elements and compute $logdet(I + \sigma\Sigma)$ on-the-fly as required. Computation of the determinant is generally performed in cubic time, whereas the computation of Σ takes $\mathcal{O}(K^2 \cdot d)$ leading to a total of $\mathcal{O}(K^3 + K^2 \cdot d)$ per element.
To improve the overall speed, we can trade some runtime with memory if we store Σ permanently. Since $k(\cdot, \cdot)$ is positive definite, $I + \sigma\Sigma$ is also positive definite. Thus, we may use a Cholesky decomposition $I + \sigma\Sigma = L^T L$ where L denotes a lower triangular matrix to compute the determinant.
Adding a new row / column vector to a Cholesky decomposition can be performed in $\mathcal{O}(K^2)$ time. Thus, when adding a new element to the summary, we compute all necessary kernel values and update the Cholesky decomposition accordingly leading to $\mathcal{O}(K^2 + K \cdot d)$ computations per element. However, with this method the memory requirements increase, since Σ and L need to be stored continuously, leading to $\mathcal{O}(2K^2 + K)$ memory instead of $\mathcal{O}(K)$ memory.

4 Experiments

In this section we experimentally evaluate the enhancements introduced to Sieve-Streaming. More specifically, we want to answer the following questions:

1. Are extracted summaries meaningful in a sense, that they represent the hidden states of a system?
2. How does the runtime decrease when a reduced number of sieves is used?
3. How does the utility value increase when dynamic thresholding is used and how does this affect the quality of the summary?

Answering the first question is difficult, because we need to know the data generation process in detail to detect the hidden states of a system. For real-world data this is nearly impossible. Hence, we use data sets $\mathcal{D} = \{(\boldsymbol{x_i}, y_i) | i = 1, \ldots, N\}$ usually found in classification tasks. Here, each observation $\boldsymbol{x}_i \in \mathbb{R}^d$ is also associated with a label y_i. We assume that each label y_i represents one hidden state of the system in the data generation process. Thus, we compute a summary of the training data $\boldsymbol{x}_i$ without considering the labels y_i and then report the number of different labels represented by the best summary found across all sieves. With respect to the notation used in classification tasks, we denote this measure as recall.

To answer the second and third question, we measure the computation time per data item and the best utility value of each algorithm (pure, reduced, and dynamic).

All experiments were performed on an Intel i7-6700 machine with 3.4 Ghz and 32 GB RAM. We repeated each experiment 10 times with shuffled data sets and always report average results. Our implementation is available at `https://bitbucket.org/sbuschjaeger/iotstreaming2017`.
Figure 3 contains all experimental results. Each column represents one data set whereas each row represents one measure. We use one synthetic data set and two real-world data sets.

Synthetic data: We composed a synthetic data set containing eight four-dimensional Gaussian distributions as a Gaussian mixture model. Each Gaussian mean value $\boldsymbol{\mu} = (\mu_i)_i$ is uniform randomly generated with $-2 \leq \mu_i \leq 3$ for $i = 1, 2, 3, 4$. Variance values are chosen uniformly random from $(0, 0.8]$.

For the data generation process we first randomly pick one of the Gaussian distributions and then sample a new data item $\boldsymbol{x} \in \mathbb{R}^4$ from this distribution. We repeat this process 10000 times. To simulate infrequent states, e.g. failure, we introduced different probabilities for each Gaussian. Two Gaussian are rare to be selected with a probability of 0.001, whereas the remaining six Gaussian receive the remaining probability mass equally, that is each has a probability of 0.133 to be used for data generation.

We use the RBF Kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2}{10})$ and set $\sigma = 1$ and $\varepsilon = 0.1$. Since there are eight hidden states in total, a summary of size $K = 8$ should be enough in theory to detect all states of the system. To account for some error however, we vary K from 10 to 24.

The first image in the first row of Figure 3 shows the recall of all three algorithms on the synthetic data set. Sieve-Streaming and its tuned counterpart achieve roughly 60% recall for $K = 10$ and then steadily increase their recall with rising K reaching nearly 100% at $K = 20$. Dynamic Sieve-Streaming shows a similar performance, but behaves slightly better for smaller K. Interestingly, for $K = 16$ all variants produce the same output and for $K = 18$ Dynamic Sieve-Streaming is outperformed by pure Sieve-Streaming. For larger K however, Dynamic-Sieve Streaming detects all states with 100% recall for $K = 24$.
Note that a potential "failure" state has a probability of 0.001, that is roughly every 1000 measurements such "failure" may occur. Usually, a human operator would need to examine all 1000 states in this case to determine if there is a failure or not. Using a summary with $K = 24$ however, a human operator only needs to examine 24 different measurments, reducing the overall workload for the human operator by 99.976%.

Looking at the second row in Figure 3, we see that Sieve-Streaming and its implementation on a smaller interval offers the same utility, which can be expected. Dynamic Sieve-Streaming consistently achieves higher utility values explaining the higher recall.

The last row of plots depict the runtime results. Sieve-Streaming needs around 0.45 ms per element, whereas its reduced variant is faster for all K. Dynamic Sieve-Streaming on the other hand needs up to 0.52 ms per element, due to its dynamic sieve management.

UJIndoor Location: As a second data set, we use the `UJIndoorLoc` data set [12]. This data set contains the GPS position and RSSI fingerprints of nearby WiFi hotspots as well as the corresponding building, floor and room number of 17 smartphone users throughout their daily routine at the university of Jaume in Spain. The dataset in total contains 19937 measurements and is usually used for prediction tasks in which one tries to infer the GPS position based on the RSSI fingerprints.

We abandon the RSSI fingerprints and try to predict the semantic location, that is the building, floor and room number directly based on the current GPS position. We normalize the GPS data and use the RBF Kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2}{0.005})$. We set $\sigma = 1$ and $\varepsilon = 0.1$. On average, each user visits 79 different locations, thus we try different K from 80 to 130.
The second column in Figure 3 shows the results for this data set. The recall is displayed in row one. Standard Sieve-Streaming and its reduced counterpart achieve between 50% and 60% recall, whereas Dynamic Sieve-Streaming manages to detect roughly 10% more items offering a recall of 60% to 70%. Again, this increase in recall can be explained with a higher utility value: Dynamic Sieve-Streaming increases the utility value consistently by roughly 10 points compared to Sieve-Streaming. The runtime of the three algorithms (second column and third row of Figure 3) paints a different picture than before. Again, pure Sieve-Streaming is the slowest algorithm, whereas Sieve-Streaming on the smaller interval reduced the overall computational costs by around 0.2ms per element. For smaller K, Dynamic Sieve-Streaming can be found in the middle of the two algorithms. For larger K however, Dynamic Sieve-Streaming seems to become faster than the reduced version of Sieve-Streaming. The reasons for that is, that Sieve-Streaming may exit a data set early if all sieves are full. Since Dynamic Sieve-Streaming reopens new sieves, it will run through the complete data set. This in turn, enables cache locality and branch prediction to take full effect, so that the runtime per element decreases.

MNIST: As a third data set, we use the well-known `MNIST` data set [6]. This data set contains 60000 28×28 grayscale images of the handwritten digits 0 to 9 and is usually used as a baseline measure for image classification. Much in line with the usual classification task, we try to extract a summary containing one representative image for each digit. However, we will do this in an unsupervised manner. Again, we normalize the data and use the RBF kernel $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-\frac{||\boldsymbol{x}_i - \boldsymbol{x}_j||_2^2}{784})$. We set $\sigma = 1$ and $\varepsilon = 0.1$. Since there are 10 classes in total, we vary K from 8 to 16.
The results are shown in the third column of figure 3. All algorithms performed nearly the same, with Dynamic Sieve-Streaming being slightly better for all values of K reaching a recall between 60 and 80%. Again, the utility value for Dynamic Sieve-Streaming is consistently larger than for pure and reduced Sieve-Streaming explaining the increased recall performance. Figure 2 displays two summaries for $K = 8$ computed by pure Sieve-Streaming and dynamic Sieve-Streaming respectively. Both algorithms show a similar summary, but pure Sieve-

Streaming selects multiple ones and fives. Dynamic Sieve-Streaming also selects multiple ones, but does not choose to include two fives into the summary and thus can add six, seven and eight to the summary increasing the recall significantly. Looking at the runtime, standard Sieve-Streaming is again the slowest algorithm variant reaching a computation time of 1ms per element, whereas reduced and dynamic Sieve-Streaming are both faster with a maximum of 0.65 ms per element. Note, that we again observe the effect, that Dynamic Sieve-Streaming seems to be faster than its reduced counterpart due to cache locality and branch prediction.

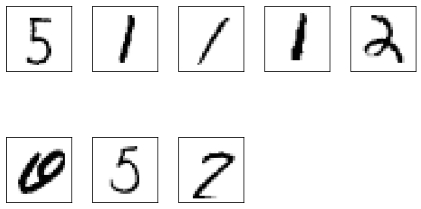
5 Conclusion

In this paper we tackled the problem of data summarization on data streams with small, embedded systems. More specifically, we tried to answer the following questions:

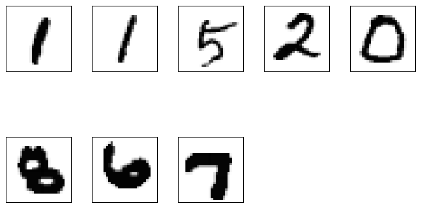
– Is submodular function maximization a suitable framework for data summarization and can summarization help us to detect the current state of a system?
– How can we make data summarization on data streams more suitable for small embedded systems frequently found in IoT settings?

In order to answer these questions, we first presented the Sieve-Streaming algorithm from [1]. Sieve-Streaming works on the assumption of submodularity and uses this to sieve out unwanted items. The main advantage of Sieve-Streaming is its strong and universal theoretical foundation offering the approximation guarantee of at least $(\frac{1}{2} - \varepsilon)$.
This theoretical foundation however, leaves room for a more specialized version of Sieve-Streaming tailored towards data summarization. Therefore, we introduced a more refined version of Sieve-Streaming for data summarization by reducing the overall number of Sieves needed. Additionally, we introduced a dynamic version of Sieve-Streaming independent from the summarization task which is able to reuse sieves with different thresholds. We note, that both enhancements do not jeopardize the theoretical analysis of Sieve-Streaming and still maintain the $(\frac{1}{2} - \varepsilon)$ approximation guarantee.
We evaluated our algorithmic changes on three data sets. The experiments have shown that we are able to detect the states of systems using the presented data summarization approach and that an increased utility function usually comes with an increased recall of system states. Thus, Dynamic Sieve-Streaming is a valuable enhancement of the standard Sieve-Streaming algorithm consistently increasing the recall leading to a 99.976% reduction of workload for human operators in our experiments.

Dynamic Sieve-Streaming has its limitations. We did not always detect all hidden states in the experiments. Future work will investigate other kernel functions, which could have been tuned more.

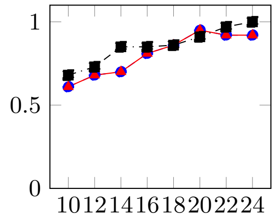


(a) Pure Sieve-Streaming

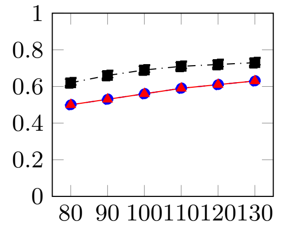


(b) Dynamic Sieve-Streaming

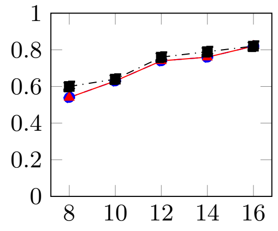
Fig. 2: Extracted summary for MNIST using pure Sieve-Streaming and Dynamic Sieve-Streaming for $K = 8$.



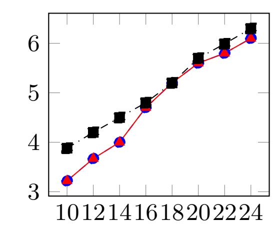
Recall synthtic data.



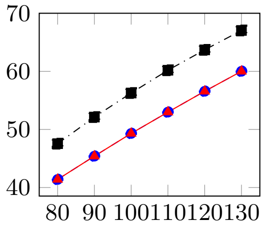
Recall `UJIndoorLoc`



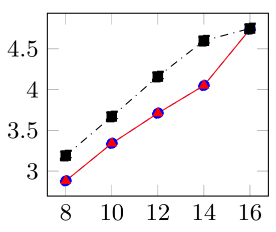
Recall `MNIST`.



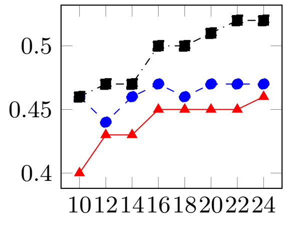
Utility synthtic data.



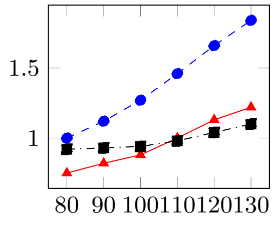
Utility `UJIndoorLoc`



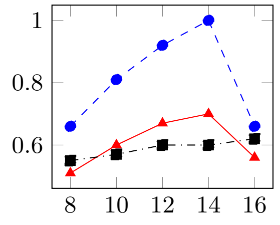
Utility `MNIST`.



Runtime synthtic data.



Runtime `UJIndoorLoc`



Runtime `MNIST`.

Fig. 3: Experiments on synthetic data (left column), UJIndoor location data (middle column) and MNIST (right column). The first row depicts recall (higher is better), the second row shows the maximum utility value (higher is better) and the third row displays the runtime (in milliseconds) per element (smaller is better) on the datasets.

Additionally, we observe that Sieve-Streaming and its newer enhancements do not deal with concept drift. Here, further work is also needed.

In sum, dynamic Sieve-Streaming is a helpful tool for exploring and monitoring data streams with a solid theoretical foundation. At the same time, we also see a potential for new research directions including concept drift and specialized kernel functions.

Notes and Comments. Part of the work on this paper has been supported by Deutsche Forschungsgemeinschaft (DFG) within the Collaborative Research Center SFB 876 (`http://sfb876.tu-dortmund.de`) "Providing Information by Resource-Constrained Analysis", project A1.

References

1. Badanidiyuru, A., Mirzasoleiman, B., Karbasi, A., Krause, A.: Streaming submodular maximization: Massive data summarization on the fly. In: ACM SIGKDD (2014)
2. Brooks, B.P.: The coefficients of the characteristic polynomial in terms of the eigenvalues and the elements of an n× n matrix. Applied mathematics letters (2006)
3. Graf, A.B., Borer, S.: Normalization in support vector machines. In: DAGM Symposium of Pattern Recognition (2001)
4. Krause, A., Golovin, D.: Submodular function maximization. In: Tractability: Practical Approaches to Hard Problems. Cambridge University Press (2014)
5. Lawrence, N., Seeger, M., Herbrich, R., et al.: Fast sparse gaussian process methods: The informative vector machine. In: NIPS (2003)
6. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE (1998)
7. Lin, H., Bilmes, J.: A class of submodular functions for document summarization. In: Meeting of the Association for Computational Linguistics (2011)
8. Mirzasoleiman, B., Badanidiyuru, A., Karbasi, A.: Fast constrained submodular maximization: Personalized data summarization. In: ICML (2016)
9. Mirzasoleiman, B., Karbasi, A., Krause, A.: Deletion-robust submodular maximization: Data summarization with "the right to be forgotten". In: ICML (2017)
10. Seeger, M.: Greedy forward selection in the informative vector machine. Tech. rep., University of California at Berkeley (2004)
11. Stolpe, M.: The internet of things: Opportunities and challenges for distributed data analysis. ACM SIGKDD Explorations Newsletter (2016)
12. Torres-Sospedra, J., Montoliu, R., Martínez-Usó, A., Avariento, J.P., Arnau, T.J., Benedito-Bordonau, M., Huerta, J.: Ujiindoorloc: A new multi-building and multi-floor database for wlan fingerprint-based indoor localization problems. In: IPIN (2014)
13. Tschiatschek, S., Iyer, R.K., Wei, H., Bilmes, J.A.: Learning mixtures of submodular functions for image collection summarization. In: NIPS (2014)
14. Wei, K., Liu, Y., Kirchhoff, K., Bilmes, J.A.: Using document summarization techniques for speech data subset selection. In: HLT-NAACL (2013)