3 Streaming Data, Small Devices

Big data often stem from sensors that stream their measurements continuously. Imagine for instance an embedded system that acts upon certain sensor inputs. *Data streams* naturally occur in the Internet of Things, embedded systems and cyber-physical systems. In particular, we encounter them in all kinds of *small devices* that have strictly limited resources like limited energy, communication bandwidth, memory, and computational power.

To model those scenarios from an algorithmic perspective and to quantify the trade-off between the required resources and the accuracy achievable by a learning algorithm, the algorithmic community has introduced the streaming model [523]. A data stream algorithm makes one pass[1] over the data, presented in N items one by one. Hereby, it maintains a summary of the stream whose size is limited to a *sublinear* amount, often polylogarithmic in N or even constant. We distinguish between different streaming models with increasingly dynamic updates:

Insertion-only data stream New items are only *added* to the data stream. The algorithm processes one item after another and views each item as a new instance.

Dynamic data stream Items can be *added* or *deleted* from the data stream. The algorithm processes the insertion or deletion of an item one after another. New items can be added to the stream or already processed items can be removed from the stream.

Turnstile data stream Every single feature or coordinate of data items can be modified by *adding* or *subtracting* values to update the current state. This is the most general case in which an algorithm needs to process the insertion of new items and updates to old items within the stream including their removal from the stream when subsequent updates add up to zero.

This chapter shows general algorithmic approaches to process and summarize streaming data and surveys recent research in this area, including several contributions of the CRC 876. It also highlights the importance of these topics for teaching so that the next generation of researchers and practitioners may tackle future challenges in this area.

Section 3.1 on summary extraction from streams presents an insertion-only data stream algorithm to maximize submodular functions, which are very important and have many applications. Prominent examples include maximizing entropy, and mutual information of selected subsets of data. The section surveys several state-of-the-art algorithms for the problem and gives an own technical contribution. It covers algorithmic and analytical methods for data streams and relaxations of worst-case conditions to

1 The number of passes is often relaxed to a small constant or logarithmic amount for problems where single pass algorithms are impossible to obtain or where a multi pass algorithm allows significantly improved results over what is possible in a single pass.

https://doi.org/10.1515/9783110785944-003

model *typical* behavior via probabilistic assumptions. The section may serve as a basis for one lecture.
Section 3.2, on coresets and sketches, introduces general concepts for summarizing data streams with respect to specific computational problems such as regression, classification, and clustering. It covers a brief technical introduction to coresets and sketches and highlights their importance for the design of data stream algorithms. It surveys the state of the art with a focus on contributions within the CRC 876. Each subsection introduces one of the main research directions and provides briefly the central ideas behind the results. The section may serve as a basis for a seminar or short lecture series on the topic.

3.1 Summary Extraction from Streams

Sebastian Buschjäger
Katharina Morik

Abstract: As processing capabilities increase, more and more data is gathered every day everywhere on earth. While machines are becoming more and more capable of dealing with these large amounts of data, humans cannot keep up with the amount of data generated every day. They need small and comprehensive representative samples of data, which capture all the informative parts of the data, in other words: a data summary. Formally, we formulate the data summarization problem as a function maximization problem with a cardinality constraint in which we seek to maximize a utility function f while selecting up to K elements in total.

Due to their compelling theoretical properties, submodular functions have been widely adopted as a utility function for data summarization. Submodular functions are set functions that reward adding a new element to a smaller set more than adding the same element to a larger set and thereby naturally lead to small and comprehensive summaries. This fits the restricted resources of small devices. We want to do a step further and model the summarization as a streaming algorithm. Streaming algorithms evaluate each data item once and decide immediately, on-the-fly with a limited memory budget, if an item should be added to the summary or not. These algorithms can be run on small, embedded devices *while* data is generated and thereby provide a data summary *anytime* with minimal computational costs.

In this contribution, we discuss the framework of submodular functions in more detail and survey the current state of the art for streaming submodular function maximization. We analyze each algorithm for performance guarantees as well as runtime and memory consumption. We end the contribution with a comprehensive comparison between algorithms for real-world summarization tasks over data streams with and without concept drift.

3.1.1 Introduction

While computers can process terabytes of data within seconds, humans are often overwhelmed with the sheer amount of information available. Humans can inspect and interact well with small, representative samples of data. Such a *data summary* must capture all the informative parts of the data while being small and comprehensive.

In recent years, submodular optimization has found its way into the toolbox of machine learning and data mining. It offers a well-established mathematical framework to select small and comprehensible summaries for a variety of different tasks. The field

of online submodular optimization studies algorithms that view each item only once and then either add it to the summary or discard it.

Exploiting submodular optimization for summarization faces algorithmic challenges right where it is needed most, namely, in the context of the Internet of Things (IoT), particularly regarding sensor networks and distributed processing, that needs to be communication-aware and energy saving. Most of the data is produced by small embedded electronics with limited processing and limited storage capabilities. Thus, a data summary should be captured *on-the-fly* while the data is being generated and before storing it. Currently, the best performing online algorithms offer an $\mathcal{O}(\frac{1}{2} - \varepsilon)$ approximation ratio where ε also influences the memory consumption of the algorithm. Even moderate choices for ε quickly result in an unmanageable resource consumption. Feldman et al. [220] showed that this approximation ratio is the best possible for data stream 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, where K is the summary size).

Existing algorithms are designed for the mathematical *worst case* and thereby have a worst-case approximation guarantee. We argue, that most practical applications are much more well-behaved. This insight allows us to move beyond the worst case and design an algorithm that delivers a good data summary under moderate assumptions. The resulting algorithm offers a probabilistic approximation ratio of $(1-\varepsilon)(1-1/\exp(1))$ with high probability $(1-\alpha)^K$, where α is the desired user certainty and K is the summary size. It performs $\mathcal{O}(1)$ function queries per item and requires $\mathcal{O}(K)$ memory. Note, that this result does not contradict the upper bound of $\mathcal{O}(\frac{1}{2}-\varepsilon)$ from [220] since our algorithm offers a better approximation quality with high probability, but not for the worst case.

In the next section we will discuss the framework of submodular function maximization. After that, we discuss existing algorithms, whereas Section 3.1.4 details the novel `ThreeSieves` algorithm. Section 3.1.5 presents practical experiments and Section 3.1.6 concludes the contribution. Parts of this text were previously published as a conference paper in [109].

3.1.2 Submodular Function Maximization over Streams

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

$$S^\star = \underset{S \subseteq D, |S|=K}{\arg\max} f(S) \tag{3.1}$$

For the empty set, we assume zero utility $f(\emptyset) = 0$. We denote the maximum of f with $OPT = f(S^\star)$. A set function can be associated with a marginal gain which represents

the increase of $f(S)$ when adding an element $e \in D$ to S:

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

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

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

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

In general, the maximization of a submodular set function is NP-hard [214], which makes solving Equation 3.1 difficult. Therefore, a natural approach is to find an approximate solution. Nemhauser et al. [230] presented a simple $(1 - (1/\exp(1))) \approx 63\,\%$ greedy approximation algorithm (denoted as `Greedy` in this contribution) for solving Equation (3.1) which runs in linear time and requires a fixed memory budget. Greedy offers a constant approximation guarantee and only requires $\mathcal{O}(K)$ memory. The disadvantage is that it requires K iterations over the entire ground set, which is costly if the ground set is very large. Moreover, multiple iterations are impossible for streaming data. Several streaming algorithms have been proposed that read each item exactly once (when D is stored on disk) or process it once on arrival (a 'true' streaming setting). An overview of these algorithms and their theoretical properties can be found in Table 3.1. It is noteworthy that the majority of these algorithms achieve a $1/2 - \varepsilon$ approximation guarantee, where ε is the desired approximation quality. A recent analysis by Feldman et al. in [220] implies that this approximation ratio is the best possible in a streaming setting and any algorithm with a better *worst-case* approximation guarantee essentially stores all the elements of the stream (up to a polynomial factor in K). Unfortunately, for all these algorithms the memory budget and the number of function evaluations per item depend on ε. Even a moderate choice of ε turns memory and runtime requirements unmanageable for small devices.

We recognize, that the worst case is often a pathological case whereas practical applications are usually much more well-behaved. Therefore, some papers recently proposed to *ignore* these pathological cases and develop algorithms with a *better* approximation guarantee in *most* cases, while using fewer function queries and less memory [109, 485, 517]. The first algorithm for monotone submodular function maximization with cardinality constraints that ignores edge cases is the `Three Sieves` algorithm proposed in [109]. It estimates the probability of finding a more informative data item on-the-fly and only adds those items to the solution that are unlikely to be 'out-valued' in the future. The resulting algorithm offers a *probabilistic* approximation ratio of $(1 - \varepsilon)(1 - 1/\exp(1)) > 1/2 - \varepsilon$ with 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.

Tab. 3.1: Algorithms for non-negative, monotone submodular function maximization with cardinality constraint K. `ThreeSieves` offers the smallest memory consumption and the smallest number of queries per element in a streaming-setting. Adapted from [109].

Algorithm	Approx. Ratio	Memory	Queries	Stream	Ref.
Greedy	$1 - 1/\exp(1)$	$\mathcal{O}(K)$	$\mathcal{O}(1)$	✗	[230]
StreamGreedy	$1/2 - \varepsilon$	$\mathcal{O}(K)$	$\mathcal{O}(K)$	✗	[258]
Random	$1/4$	$\mathcal{O}(K)$	$\mathcal{O}(1)$	✓	[215]
PreemptionStreaming	$1/4$	$\mathcal{O}(K)$	$\mathcal{O}(K)$	✓	[91]
IndependentSetImprovement	$1/4$	$\mathcal{O}(K)$	$\mathcal{O}(1)$	✓	[121]
SieveStreaming	$1/2 - \varepsilon$	$\mathcal{O}(K \log(K)/\varepsilon)$	$\mathcal{O}(\log(K)/\varepsilon)$	✓	[32]
SieveStreaming++	$1/2 - \varepsilon$	$\mathcal{O}(K/\varepsilon)$	$\mathcal{O}(\log(K)/\varepsilon)$	✓	[367]
Salsa	$1/2 - \varepsilon$	$\mathcal{O}(K \log(K)/\varepsilon)$	$\mathcal{O}(\log(K)/\varepsilon)$	(✓)	[540]
ThreeSieves	**$(1-\varepsilon)(1-1/\exp(1))$ with prob. $(1-\alpha)^K$**	$\mathcal{O}(K)$	$\mathcal{O}(1)$	✓	**[109]**

3.1.3 Related Work

For a general introduction to submodular function maximization, we refer interested readers to [393] and for a more thorough introduction into the topic of streaming submodular function maximization to [124]. Most relevant to this contribution are non-negative, monotone submodular streaming algorithms with cardinality constraints. To the best of our knowledge, there exist six different algorithms which we survey here. The theoretical properties of each algorithm are summarized in Table 3.1.

While not a streaming algorithm, the `Greedy` algorithm [230] forms the basis of many algorithms. It iterates K times over the entire dataset and greedily selects the 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` [258] is its adaption to streaming data. It replaces an element in the current summary if it improves the current solution by at least v. It offers an $1/2 - \varepsilon$ 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 the Appendix of [32] for an example). We therefore consider `StreamGreedy` not to be a real streaming algorithm. Similar to `StreamGreedy`, `PreemptionStreaming` [91] compares each marginal gain against a threshold $v(\mathcal{S})$. Here, 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 $1/4$. Feige et al. show in [215] that for any non-negative submodular function a uniformly chosen random set is a $1/4$ approximation. A uniform random set over a data stream can be obtained via reservoir sampling [688]. Also Chakrabarti and Kale proposed in [121] a streaming algorithm 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. [540] propose 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. Their algorithm offers a $1/2 - \varepsilon$ approximation, but some of the thresholding rules require additional information about the data stream such as its length or density. Since this is unknown in a true streaming setting, this algorithm is not completely streaming-capable.

The first real streaming algorithm with $1/2 - \varepsilon$ approximation guarantee was proposed by Badanidiyuru et al. [32] 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 let $|S| < K$, then an element e is added to the summary S if the following holds:

$$\Delta_f(e|S) \geq \frac{OPT/2 - f(S)}{K - |S|} \tag{3.2}$$

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 $m = \max_{e \in D} f(\{e\})$ 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 the management of 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 $m = \max_{e \in D} f(\{e\})$ 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 [367] 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 leads to an algorithm in which sieves are removed once they are outperformed by other sieves and new sieves are introduced to leverage the better estimation of OPT. `SieveStreaming++` does not improve the approximation guarantee of `SieveStreaming`, but only requires $\mathcal{O}(K/\varepsilon)$ memory instead of $\mathcal{O}(K \log K/\varepsilon)$.

3.1.4 Getting More by Doing Less

`SieveStreaming` and its extension offer a worst-case guarantee on their performance and indeed they can be considered optimal, providing that there is an approximation guarantee of $1/2 - \varepsilon$ under polynomial memory constraints in ε and K [220]. 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 it follows the same distribution, e.g. for a certain time frame. In this contribution, we want to investigate these better behaving cases carefully. This allows us to present an algorithm that 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. From a data streams perspective this assumption means, that we ignore concept drifts and assume that an appropriate *concept drift detection* mechanism is in place, so that summaries are, e.g., re-selected periodically. For batch processing this means, that all items in the batch should come from the same (yet unknown) distribution. Please note, that in this case we do *not* assume that all possible samples come from the same distribution, but we merely assume that the sample we are given is consistent in the sense that all items come from *the same* distribution. This is true for *all* data samples, where data items are independent from each other, as we could simply define the overall distribution as a mixture of simpler distributions. We now use this assumption to derive an algorithm with $(1 - \varepsilon)(1 - 1/\exp(1))$ approximation guarantee of high probability.

`SieveStreaming` and its extension maintain $\mathcal{O}(\log(K)/\varepsilon)$ sieves in parallel, which quickly becomes unmanageable even for moderate choices of K and ε. Both algorithms show the following behavior: many sieves in `SieveStreaming` quickly fill-up with uninteresting events if their novelty threshold is *too small*. `SieveStreaming++` exploits this observation by removing small thresholds early on and focuses on the most promising sieves in the stream. If the novelty threshold is *too large*, both algorithms deliver sieves that never include any item. Actually, there are only a few thresholds that produce small and comprehensive summaries.

The management of many sieves, each with its own threshold might be unnecessary. Instead of using many sieves with different thresholds we use only a single summary and carefully calibrate the threshold: 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.

As discussed, the set $O = \{(1 + \varepsilon)^i \mid i \in \mathbb{Z}, m \leq (1 + \varepsilon)^i \leq K \cdot m\}$ offers a sufficient approximation of OPT. We start with the largest threshold in O and decide for each item whether we want to add it to the summary or not. If we do not add any of these items (the exact threshold T for this will be discussed later) to S we may lower the threshold to the next smaller value in O and repeat the process until S is full.

The key question now becomes: How to choose the threshold T appropriately? If T is too small, we will quickly fill up the summary before any interesting items arrive that would have exceeded the original threshold. If T is too large, we may reject interesting items. 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 a bounded 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, i.e.,

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

we can safely lower the threshold. Now, we have transformed the original problem of choosing the right threshold of utility to that of choosing the right length of T and arrive at the problem of estimating the probability of making the right choice. Moreover, 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 [81]. However, there exists a more direct way of computing a confidence interval for heavily one-sided binominal distribution with probabilities near zero [351] when the novelty of items is independent and identically distributed (iid). Then, the probability of not adding one item in T trials is:

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

A first order Taylor approximation of $\ln(1 - p(e|f, S, v))$ reveals that

$$\ln\big(1 - p(e|f, S, v)\big) \approx -p(e|f, S, v)$$

and therefore $\ln(\alpha) \approx T(-p(e|f, S, v))$ leading to:

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

Hence, the confidence interval of $p(e|f, S, v)$ after observing T events is $\left[0, \frac{-\ln(\alpha)}{T}\right]$. For example, with 95 % certainty the confidence interval of $p(e|f, S, v)$ is $\left[0, -\ln(0.05)/T\right]$ which is approximately $[0, 3/T]$ leading to the term *Rule of Three* for this estimate [351]. We can use the Rule of Three to quantify the certainty that with high probability there will not be a novel item in the data stream after observing T items.

Note, that we can set α and the user-defined threshold τ and then compute the minimum of the required number of observations T with the above relationship. Alternatively, we may directly specify the maximum number of observations T as a user parameter instead of α and τ, thus removing one hyperparameter. We call our algorithm

ThreeSieves due to its usage of the Rule of Three. It is depicted in Algorithm 1 and analyzed theoretically in Theorem 1.

Algorithm 1: ThreeSieves algorithm.

$O \leftarrow \{(1+\varepsilon)^i \mid i \in \mathbb{Z}, m \leq (1+\varepsilon)^i \leq K \cdot m\}$
$v \leftarrow \max(O)$; $O \leftarrow O \setminus \{\max(O)\}$
$S \leftarrow \emptyset$; $t \leftarrow 0$
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**
      $v \leftarrow \max(O)$
      $O \leftarrow O \setminus \{\max(O)\}$
      $t \leftarrow 0$

Theorem 1. ThreeSieves *has the following properties [109]:*

- *Given a fixed groundset D or an infinite data stream in which each item is independent and identically distributed (iid),*
- ThreeSieves *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: $f(S) \geq (1-\varepsilon)(1-1/\exp(1))OPT$.*
- *It does one pass over the data (truly streaming) and stores at most $\mathcal{O}(K)$ elements.*

3.1.5 Experiments

We now experimentally evaluate the following four questions:

Q1 Is ThreeSieves' performance competitive against the other algorithms in its practical performance given its probabilistic guarantee?
Q2 If ThreeSieves is competitive, how does it relate to a uniform random selection of summaries?
Q3 How does ThreeSieves behave for different T and different ε?
Q4 How large is the resource consumption of ThreeSieves in comparison with the other algorithms?

We ask each algorithm to select a summary with exactly K elements. Since most algorithms can reject items, they may select a summary with fewer than K elements. This

makes a comparison between different algorithms difficult, because it favors algorithms with larger summaries (f is monotone and hence adding items to the summary *always* increases the function value), but not necessarily better summaries. For a fair comparison we ensure that all algorithms select a summary of size K by re-iterating over the entire dataset 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. We also measure the 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.

We will focus on two real-world data-sets. First, the ForestCover [157] data-sets contains 286 048 examples of different forest cover types. Forest cover is the amount of land area that is covered by forest. This proportion is structured into classes. The learning task for this data-set is to predict the class of each cover by using the 10 provided cartographic variables that are obtained via remote sensing. Second, the Creditfraud [443] data-set contains 284 807 fraudulent and legal bank transactions. The learning task for this data-set is to classify each transaction using their 29 features. However, we are interested to see a data summary for a user' manual inspection of the data. Hence, in our experiments we ignore the class information and aim at selecting a diverse set of examples based on the features. More experiments using the novel `ThreeSieves` algorithm can be found in [109].

We extract summaries of varying sizes $K \in \{5, 10, \dots, 100\}$ maximizing the log-determinant

$$f(S) = \frac{1}{2} \log\det(\mathbb{I} + a\Sigma_S). \tag{3.3}$$

$\Sigma_S = [k(e_i, e_j)]_{ij}$ is a kernel matrix containing all similarity pairs of all points in S, $a \in \mathbb{R}_+$ is a scaling parameter, and $\mathbb{I}$ is the identity matrix. In [619], this function is shown to be submodular. Its function value does not depend on the ground-set D, but only on the summary S, which makes it an ideal candidate for summarizing data in a streaming setting. In [111], it is proven that $m = \max_{e \in D} f(\{e\}) = 1 + aK$ 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 [268]. 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 $T \in \{500, 1000, 2500, 5000\}$. [2]

We present two different sets of plots. Figure 3.1 contains plots for varying K with a fixed $\varepsilon = 0.001$ (top figure) and plots for varying ε with fixed $K = 50$ (bottom plot). Both figures show the relative performance, the runtime and the memory consumption for different algorithms. Note, that we excluded `Random`, `IndependentSetImprovement`, and `Greedy` for varying ε as their performance is independent of it.

2 The code for these experiments is available under https://github.com/sbuschjaeger/SubmodularStreamingMaximization/.

Performance over Different K 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, yet Salsa and SieveStreaming performing best. Using $T \leq 1000$ for ThreeSieves seems to decrease the performance, which is reflected by the weaker guarantee of the algorithm. On Creditfraud, ThreeSieves performs *better* than Greedy with a relative performance above 100. Note, that only ThreeSieves showed this behavior, whereas the other algorithms never exceeded Greedy. As expected, a uniform random selection shows the weakest performance. SieveStreaming and SieveStreaming++ show identical behavior.

Please, note the logarithmic scale of the runtime. Here, we see that ThreeSieves and Random are by far the fastest methods. Using $T = 1000$ offers some performance benefit, but it 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 least resources as our algorithm only stores a single summary in all configurations. These curves are identical with Random so that only 4 instead of 7 curves can be seen. SieveStreaming and their siblings use roughly two magnitudes more memory since they keep track of multiple sieves in parallel. As expected, SieveStreaming++ uses less memory than SieveStreaming which uses less memory than Salsa.

Performance over Different ε The behavior of the algorithms for different approximation ratios shows a slightly different picture than before. For larger $\varepsilon > 0.05$ the performance of the non-probabilistic algorithms remain relatively stable, but ThreeSieves performance starts to deteriorate. For small $\varepsilon \leq 0.05$ and larger T ThreeSieves and Salsa again perform best in all cases. Again, SieveStreaming and SieveStreaming++ show identical behavior. Regarding 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. Again, note, that ThreeSieves requires the same amount of memory in all configurations and hence we find an overlap in plots.

We Conclude the Experiments In summary, ThreeSieves is competitive to the other algorithms and sometimes even outperforms them. The probabilistic guarantee of the algorithm comes along with a competitive performance in many cases while using fewer resources. In some cases ThreeSieves even outperforms the Greedy algorithm. ThreeSieves works best for small ε and large T. In contrast to the other algorithms, the resource consumption and overall runtime of ThreeSieves does not suffer from decreasing ε or increasing T.

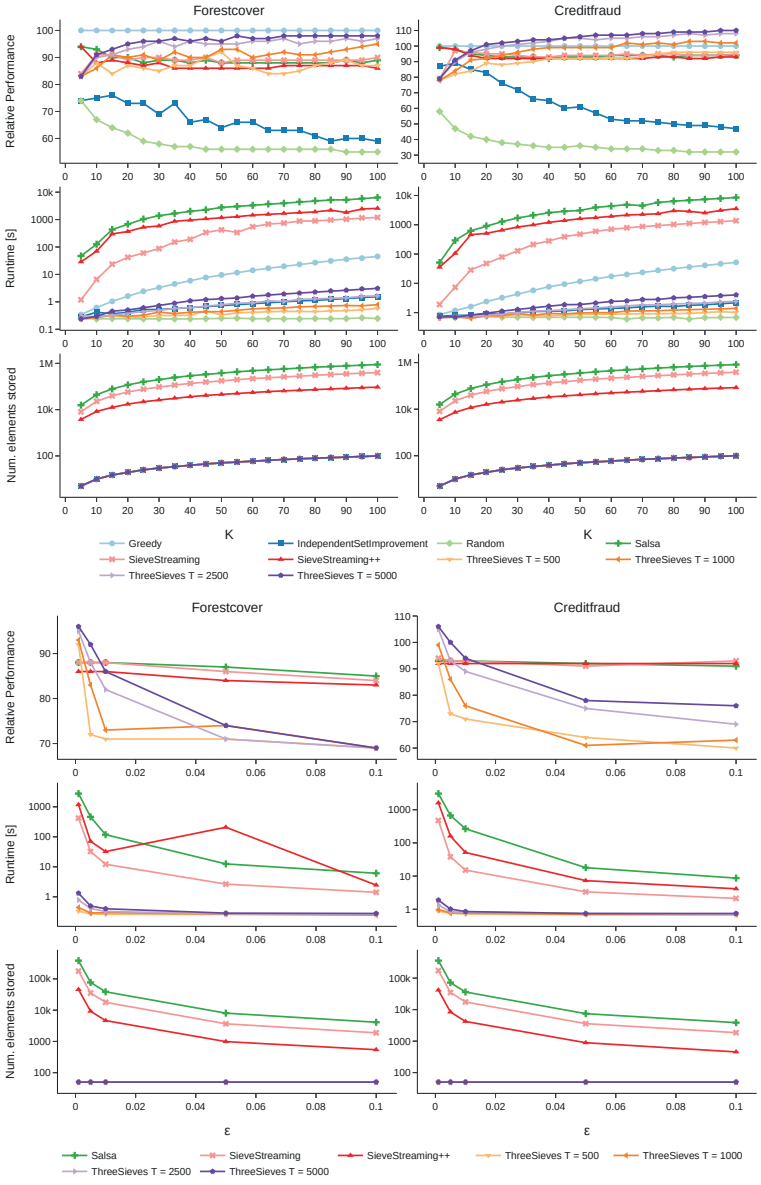

Fig. 3.1: Comparison of `IndependentSetImprovement`, `SieveStreaming`, `SieveStreaming++`, `Salsa`, `Random`, and `ThreeSieves` for different K values with fixed $\varepsilon = 0.001$ (top figure) and different ε with fixed $K = 50$ (bottom figure). The first row shows the relative performance to `Greedy` (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 data-set.

3.1.6 Conclusion

Data summarization is a valuable tool for humans to inspect and understand large amounts of data at a quick glance. For complex and long running processes these summaries must be selected online while the data generating process takes place. While the quality of a summary can be highly subjective to the task and person, submodular functions offer a well-established mathematical framework to produce small and comprehensible summaries for a variety of different tasks. In this Section, we discussed submodular functions and their maximization for data summarization. We focused on the task of stream summarization in which each item is evaluated only once and it must be decided on-the-fly whether it should be added to the summary or not. We reviewed existing algorithms and their theoretical properties in this realm. They are optimized towards the worst-case, whereas practical problems are often much more well-behaved, in particular the data inside the stream are most often independent and identically distributed (iid). This allows the `ThreeSieves` algorithm to compute good summaries with high probability. We experimentally showed that `ThreeSieves` not only outperforms the current state of the art, but also uses fewer resources by an order of magnitude. The algorithm is designed such that kernel functions can be chosen. This enables a more interactive data exploration for the human user, by, say, reviewing multiple summaries with different kernel functions in a very short period of time.

3.2 Coresets and Sketches for Regression Problems on Data Streams and Distributed Data

Alexander Munteanu

Abstract: Coresets and sketches are small data summaries for a given computational problem such as regression or clustering. They preserve the cost function for any possible solution up to little distortion and thus serve as a proxy for the original massive dataset during optimization or inference. They have strong aggregation properties such as linearity or mergeability and thus facilitate their construction for data streams as well as for distributed data. Once the data summary is computed, it can be analyzed using a classical algorithm and the result will be provably close to an optimal solution. In summary, this improves the efficiency and scalability and enables streaming and distributed computation using standard offline algorithms.

We show how linear sketching enables streaming and distributed data processing and show how even static off-line coreset constructions can be extended to those flexible computational settings via the Merge & Reduce principle. Next we survey classic sketching and coreset results for ordinary linear regression and show how those can be extended to more sophisticated models, such as Bayesian regression, generalized linear models, and dependency networks. We also show the limitations of data summarization via complementing lower bounds and how natural assumptions and parameterized beyond-worst-case analysis help to overcome those limitations.

3.2.1 Introduction

Developing highly efficient regression approaches is an important research direction that aims at making modern statistical regression methods scalable to large and high-dimensional data and also to settings where computational resources are scarce as is often the case in the Internet of Things (IoT). We pursue this goal via modern data reduction approaches: we have seen in Section 3.1 how direct sampling methods can summarize the items presented in a data stream. Here we focus on two further methods called *sketches* and *coresets*. Those three approaches are arguably the most promising and widely used methods to facilitate the analysis of mass data with provable accuracy guarantees. See [565] for an extensive survey. In recent years a new paradigm called *sketch-and-solve* has been established for dealing with mass data. The idea behind sketch-and-solve is to apply a simple and fast dimensionality reduction technique in a first step to compress the data to a significantly smaller *sketch* of at most polylogarithmic size. Next, as a second step, we feed the sketch as input to a standard solver for the problem. The theoretical challenge is to prove an approximation guarantee for the

solution obtained from the sketch with respect to the original massively large dataset. The general algorithmic principle is shown in the following scheme:

$$\begin{array}{ccc} X & \xrightarrow{\Pi} & \Pi(X) \\ \downarrow & & \downarrow \\ f(\beta|X) & \approx_{\varepsilon} & f(\beta|\Pi(X)). \end{array}$$

The classical way of data analysis is indicated by the left path, where we would feed the massive dataset X directly to the algorithm and perform the computationally demanding data analysis or learning task indicated by $f(\beta|X)$. This might not even be possible when the data does not fit in main memory or we encounter other computational restrictions. Instead, we follow the path to the right, where the massive dataset X is reduced via a dimensionality reduction mapping Π to obtain a significantly smaller data summary $\Pi(X)$ that is simple to calculate. The latter now fits into main memory and can be given as input to the classical algorithm for an efficient data analysis. The bottom line indicates that the result from analyzing the massive data is close to the result obtained from the sketch. A comprehensive example is given in [245] where 2 TB of data are compressed to only 140 MB with a parameter estimation error of less than 4×10^{-6} for a streamed Bayesian linear regression task.

In light of the sketch-and-solve paradigm, we focus on algorithmic approaches for the data reduction Π that can be efficiently implemented in streaming settings as well as in distributed environments. In particular, we develop methods to aggregate data and to reduce the number of observations using sketches via random linear projections and coresets obtained by importance sampling.

Sketching and coreset methods for regression on large-scale data are important areas of research with many interesting open questions. Although basic models are meanwhile well understood, research on more complex modern statistical and machine learning methods has just begun.

3.2.1.1 Brief Introduction to Coresets

Coresets are small, possibly weighted datasets that are designed to approximate an input dataset with respect to a computational problem. A survey on common techniques for obtaining coresets is given in [516]. Coresets usually depend on the considered objective function or on a broader class of objective functions. The first definitions were only implicitly given or were restricted to specific problems such as shape fitting or clustering [35, 295]. We give a more general definition.

Definition 2 (see [516]). *Let X be a set of points from a universe U and let Γ be a set of candidate solutions. Let $f : U \times \Gamma \to \mathbb{R}^{\geq 0}$ be a non-negative loss function. Then a set $C \subseteq X$ is an ε-coreset of X for f and some $\varepsilon \geq 0$, if*

$$\forall \gamma \in \Gamma : |f(X, \gamma) - f(C, \gamma)| \leq \varepsilon \cdot f(X, \gamma).$$

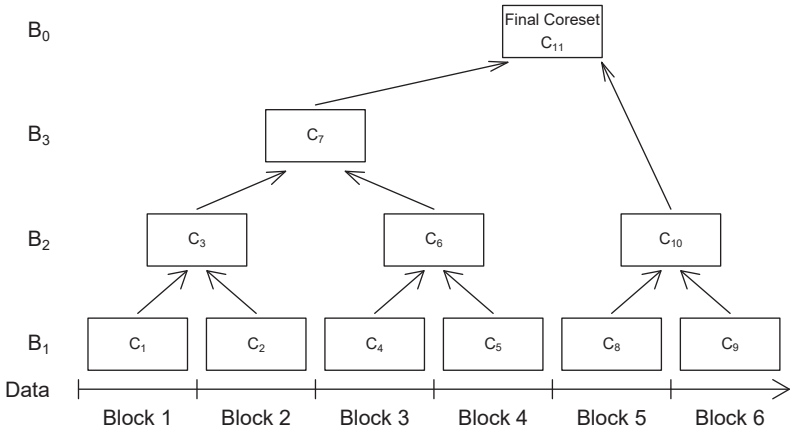

Fig. 3.2: Illustration of the Merge & Reduce principle from [246].

Note that the original point set is a perfectly accurate 0-coreset but has linear size. To be a useful data reduction, a coreset is required to be of sublinear size, e.g., polylogarithmic or even constant in the number of input points. The dependence on their dimension is usually allowed to be a small polynomial.

Coresets have been studied extensively for nearly two decades as a data aggregation and reduction tool to address scalability issues for a plethora of computational problems. Coresets have been developed for shape-fitting problems [6, 7, 33, 34, 218, 402], clustering [35, 216, 219, 453, 485], classification [294, 297, 594], ℓ_2-regression [147, 188, 189, 432], ℓ_1-regression [141, 142, 637], ℓ_p-regression [161, 709], M-estimators [144, 146], generalized linear models, [327, 501, 517, 594, 664], and other areas. We refer to [565] for an extensive survey and to [516] for a technical introduction to coresets.

Aggregation Properties and Merge & Reduce Most coreset constructions have strong aggregation properties as outlined in [295] for instance:

Definition 3. *Coresets are called* mergeable *if they satisfy the following properties:*

1. *If C_1 and C_2 are ε-coresets for input sets P_1 and P_2 respectively, then $C_1 \cup C_2$ is an ε-coreset for $P_1 \cup P_2$.*
2. *If C_1 is an ε-coreset for C_2, and C_2 is a δ-coreset for C_3, then C_1 is an $(\varepsilon + \delta)$-coreset for C_3.*

Given an off-line coreset construction that satisfies those properties, we can easily process data streams and distributed (or parallel) data via *Merge & Reduce* as a black box technique. Merge & Reduce was first introduced in [47] as a general method for

extending static data structures to handle insertions. More recently, it has been adapted to work on coresets in the streaming setting [6, 295]. Nowadays, it is one of the main tools in the design of efficient streaming and distributed algorithms for the analysis of mass data. Though often only implicitly mentioned, Merge & Reduce has become a standard technique in the coreset literature. The **merge**(C_1, C_2) operation simply takes the union as in the first item of Definition 3 whereas the **reduce**(P) operation calls the off-line coreset construction algorithm on the point set P which can be used recursively to compute an ε-coreset from an ε-coreset etc. using the second item of the definition. Hereby, the error accumulates to εk after k recursive applications so one should control the value of $k = O(\log n)$ by, say, employing a binary tree construction as in Figure 3.2.

Figure 3.2 illustrates the principle of Merge & Reduce data stream algorithms. Note that all coresets are numbered in the order in which they are generated in a sequential data streaming application. First, Block 1 containing a fixed number of points is read from the stream into memory and the coreset C_1 is calculated. The same process yields coreset C_2 derived from the data contained in Block 2 of the stream. Since C_1, C_2 are siblings in the tree, they are combined into $C_3 :=$ **reduce**(**merge**(C_1, C_2)). At this point C_1, C_2 are not needed any more and are thus deleted from memory. The Merge & Reduce operations are indicated by the arrows in Figure 3.2. Now we proceed with C_4 derived from Block 3 and C_5 derived from Block 4. Since C_4, C_5 are siblings in the tree, they are combined into $C_6 :=$ **reduce**(**merge**(C_4, C_5)) and deleted. Again we have siblings C_3, C_6 on the same level, which are combined to $C_7 :=$ **reduce**(**merge**(C_3, C_6)) and deleted. The procedure is continued in the same manner until we reach the end of the stream. Say this is the case after processing Block 6. Note that at this point C_8, C_9 have been merged and reduced into C_{10} and have been deleted. The current state of the data structure is that it holds only coresets C_{10} in bucket B_2, i.e., on level 2, and C_7 in bucket B_3, i.e., on level 3, respectively. The buckets B_0 and B_1 are empty at this point and there are no further levels above level 3. Now a postprocessing step implicitly merges $C_{11} =$ **reduce**(**merge**(C_7, C_{10})) via the working bucket B_0.

The construction can also be computed in a parallel or distributed setting. One possible scheme to achieve this, is to compute all coresets on the same level in parallel, starting with coresets C_1, C_2 C_4, C_5, C_8, C_9 on level 1 and proceeding with parallel computation of C_3, C_6, C_{10} on level 2 followed by C_7 on level 3 and finally deriving the final coreset C_{11} from C_7 and C_{10}.

Techniques that are similar to Merge & Reduce were employed in the area of physical design for relational databases [88]. Another interesting variant of Merge & Reduce directly combines statistical models rather than data summaries such as coresets [246]. We refer to Section 2.4.3 in Volume 3 of this book series for details.

3.2.1.2 Brief Introduction to Sketches

Sketching was introduced in the context of the theory of streaming algorithms. Popular examples include the `Count`-Sketch [123] the `CountMin`-Sketch [154], and the `Rademacher`-

Sketch [145]. Many contemporary sketches are variations or descendants of those basic techniques; see [708] for a survey and technical introduction. Similar to a coreset, a sketch is a succinct data summary, but it is not restricted to a subsample of the input or to representative substitute data points. Instead, any data structure of sublinear size with an efficient update procedure for processing new points may be regarded as a sketch. Usually, one encounters linear mappings, i.e., sketching matrices in the literature. Indeed, most known data stream algorithms are represented by linear sketches and there is some evidence that linear sketches are nearly optimal for such algorithms under certain conditions [429]. Linear sketches can be maintained dynamically in a data stream. Also, they have strong aggregation properties, which allow the combination of individual sketches—stemming from distributed data—to one single sketch for the entirety of the data. Sketching methods are much better positioned than coresets for handling high velocity streams, as well as highly unstructured massive databases [249, 628] and arbitrarily distributed data [648]. Linear sketches allow efficient applications in single pass sequential streaming and in distributed environments, see. e.g., [145, 354, 709]. Both, streaming and distributed computational settings are fundamental in the analysis of very large datasets and are very important for embedded systems and cyber-physical systems.

Linear sketches can be updated in the most dynamic streaming setting, which is commonly referred as the *turnstile* model, cf. [523]. In this model, we initialize a matrix X to be the all-zero matrix. The stream consists of $(key, value)$ updates of the form (i, j, v), meaning that X_{ij} will be updated to $X_{ij} + v$. Any entry can be defined by a single update or by a subsequence of not necessarily consecutive updates. For instance, the sequence $\ldots, (i, j, 25), \ldots, (i, j, -7), \ldots$ will result in $X_{ij} = 18$. Deletions are possible by using negative updates matching previous insertions. Due to linearity, linear sketches support operations such as adding, subtracting, and scaling entire databases X_j (i.e., matrices or vectors) efficiently in the sketch space, since $\Pi X = \Pi \sum_j \alpha_j X_j = \sum_j \alpha_j \Pi X_j$. For instance, if X_{t_1} and X_{t_2} are balances of bank accounts at time steps $t_1 < t_2$, then $\Pi T = \Pi X_{t_2} - \Pi X_{t_1}$ is a sketch of the transactions T within the period $t \in (t_1, t_2]$.

3.2.2 Our Contributions

Our research focused on developing streaming algorithms for frequentist and Bayesian linear regression as well as for generalized linear regression models. A common theme consists in developing data reduction techniques such as sketching via random linear projections or coresets via importance subsampling, retaining the statistical information up to little distortion. Hereby, we address resource restrictions such as memory access, communication cost, and runtime. Some highlights developed in the CRC 876 include coresets for specific classes of generalized linear models [501, 513, 515, 517] as well as graphical models [501]. We developed sketches for Bayesian linear regression models [245] and extended them towards hierarchical priors [247] and generalized normal

distributions defined over ℓ_p-spaces [511, 513, 637]. We translated the Merge & Reduce principle from data summaries to maintaining statistical summaries in the streaming model [246] and introduced an asymptotic data stream model [303]. Another significant contribution lies in a dimensionality reduction for high-dimensional Bayesian optimization in sketching-based embeddings of low-dimensional subspaces [512]. An interesting further research direction is the development of dimensionality reduction techniques for reducing the width of neural networks and studying the limitations thereof [514].

3.2.2.1 Streaming Algorithms for Generalized Linear Regression

Generalized linear models (GLMs) extend classical linear regression to more flexible classes of generating distributions, cf. [479]. Usually, one assumes that the realizations of the dependent variable are generated from a member of the exponential family of distributions, based on the independent observations. Well-known examples of such distributions include the normal, binomial, Poisson, and gamma distributions. The expectation of the dependent target variable Y is connected to the linear predictor $X\beta$ via a link function h,

$$h(\mathbb{E}(Y)) = X\beta,$$

where X is the independent feature variable and β is the unknown parameter vector.

There is extensive work on sampling methods for approximating regression problems including ℓ_2-regression [188, 189] and ℓ_1-regression [141, 142, 637]. Those were generalized to ℓ_p-regression for all $p \in [1, \infty)$ [161, 709]. More recent works studied sampling methods for M-estimators [143, 144, 146] and generalized linear models [327]. We continued this line of research on coresets and sketches for logistic regression [515, 517] and p-generalized probit regression [513].

Logistic Regression Logistic regression is an important instance of a Generalized Linear Model [479]. The aim of logistic regression is to estimate the parameter β implicitly defining Bernoulli distributions based on the observed data. An exemplary task would be to assess the impact and interactions of variables in predicting the probability of patients suffering from a certain disease, based on their personal, physiological, and diagnostic data. This learning task is based on a fixed set of patient data $X \in \mathbb{R}^{n \times d}$ and corresponding labels $Y \in \{-1, +1\}^n$ indicating whether a patient is healthy or not. Folding the labels into the data we define row-vectors $Z_i = Y_i X_i$ for all $1 \leq i \leq n$.

Our first result in [517] shows the impossibility of compressing the data sublinearly in the input size, which holds in the worst case for any data reduction technique. To get around this limitation, we introduced a novel parameter that can be used to bound the complexity of compressing a dataset Z for logistic regression. This parameter is defined by

$$\mu(Z) = \sup_{\beta \in \mathbb{R}^d \setminus \{0\}} \frac{\|(Z\beta)^+\|_1}{\|(Z\beta)^-\|_1},$$

where $(Z\beta)^+$, $(Z\beta)^-$ comprise only the positive and negative entries of $Z\beta$, respectively. We call a dataset μ-complex if it satisfies $\mu(Z) \leq \mu$. If the data is μ-complex for a small, not necessarily constant μ, then there exists an importance sampling and reweighting scheme based on the sensitivity framework of [219, 416] that produces an ε-coreset of sublinear size $O(\varepsilon^{-2}\mu\sqrt{n}d^{3/2}\log^{O(1)}(\mu nd))$ with high probability. A more involved recursive sampling scheme produces an ε-coreset of size $O(\varepsilon^{-4}\mu^3 d^3\log^{O(1)}(\mu nd))$, which is beneficial if the data is well-behaved and the input size is particularly large. Those are the first provably sublinear coreset constructions for logistic regression.

The parameter $\mu(Z)$ has an intuitive statistical interpretation and might be of independent interest as detailed in [517]. It is not uncommon in practice that $\mu(Z)$ is small, since otherwise logistic regression exhibits methodological weaknesses.

Our experimental evaluation in [517] on real-world benchmark data shows that there is an efficient implementation based on a sketched QR-decomposition that is more accurate than uniform random sampling and state-of-the-art heuristic approaches such as described in [327] while being competitive in terms of runtime.

Meanwhile, the coreset size has been reduced to $\tilde{O}(\varepsilon^{-2}\mu^2 d)$ by replacing the leverage scores in the importance sampling distribution by ℓ_1-Lewis weights [462]. This has also improved the accuracy in experiments slightly, albeit at the cost of an increased runtime.

However, one limitation of known coreset constructions is that they require two passes over the data, one for approximating the importance sampling distribution and another for subsampling and collecting the data. The recursive improvement to polylogarithmic size, or calculating Lewis weights, requires even $O(\log\log n)$ passes. The Merge & Reduce framework is no remedy here due to the assumption of a small $\mu(Z)$. One might argue that a random order stream satisfies this condition for every batch of data, but in a worst-case setting we would have $\mu(Z_i) \to \infty$ for some batch Z_i, even in cases where $\mu(Z) = 1$.

Sketching Logistic Regression Towards creating a single-pass turnstile streaming algorithm for mild μ-complex data with all computational flexibilities, we developed the first linear sketch for logistic regression. Our main result [515] is a distribution over stacked sparse random matrices

$$\Pi = \begin{bmatrix} S_0 \\ S_1 \\ \vdots \\ S_{O(\log n)} \\ T \end{bmatrix}$$

Here, at each level i, S_i first subsamples a 2^{-i} fraction of the input points which are then hashed into a small number of buckets, where collisions are handled by summing the elements in the same bucket. The construction is complemented by a small uniform

sampling matrix T. The resulting sketch reduces n input points in d dimensions to only $O(\text{poly}(\mu d \log n)) \times d$. We prove that ΠZ can be calculated over a turnstile stream in input sparsity time, i.e., $O(1)$ is spent on each non-zero element of the input. Moreover, with high probability over the random construction of Π, we have for $\tilde{\beta} \in \text{argmin}_\beta f(\Pi Z \beta)$ that

$$f(Z\tilde{\beta}) \leq O(1) \min_{\beta \in \mathbb{R}^d} f(Z\beta),$$

where f denotes the logistic loss function [515]. The intuition behind this approach is that coordinates are grouped according to *weight classes* of similar loss that can be handled separately in the analysis. Weight classes with a small number of members will be approximated well on sketching levels with a large number of elements since roughly all members need to be subsampled to obtain a good estimate. Weight classes with many members will be approximated well on levels with a smaller number of subsamples. This is because if too many members survive the subsampling there will also be too many collisions under the uniform hashing, which would either lead to a large overestimate when those add up, or, due to asymmetry, would cancel each other and lead to large underestimations. Dealing with the asymmetry of the logistic loss was another issue that needed to be controlled. The error could not be bounded if the sign of an element was confused, since the ratio $\ell(x)/\ell(-x)$ is unbounded for the loss function $\ell(\cdot)$ of unconstrained logistic regression. Finally, there could be too many small contributions near zero. Logistic regression, unlike norms, assigns a non-zero constant loss to them. Their contribution can thus become significant. This is taken care of by the small uniform sample T of size $\tilde{O}(\mu d)$.

Poisson Regression Poisson regression is another instance of a GLM model, which aims at modeling count variables [479, 706]. A prominent example within the CRC 876 can be found in Section 4.1 in Volume 3 of this book series, where Poisson models are used to predict the number of vehicles per minute passing sensors of the highway ring around the city of Cologne. The predictions for a single sensor location are made based on the measurements at all other locations and the parameters learned from a Poisson regression model [284, 286]. This can be formalized as a Poisson dependency network (DN) [301]. Dependency networks are graphical models comprising a collection of GLMs, where each element of a set of d variables is regressed on all other variables. Dependency networks have several interesting applications surveyed in [501], such as collaborative filtering and density estimation, phylogenetic analysis, genetic analysis, network inference from sequencing data, and traffic modeling as well as topic modeling.

In our work [501], we have developed coresets for dependency networks. Assuming all GLMs in the dependency network to be ordinary linear regression models, we can subsample and reweight the input points as in [188] to construct a coreset. Surprisingly, we do not need to construct a coreset for each of the d GLMs separately. Instead, we

can exploit the common subspace structure of all GLMs to show that it is sufficient to construct one single coreset of size $O(\varepsilon^{-2} d \log d)$.

With Poisson GLMs, the situation is different. Again, we can show that in the worst case, any data reduction technique produces either a summary of linear size or fails to approximate the objective function to within a large superconstant factor [501]. Reviewing the statistical modeling for count data, we note that the Poisson lognormal model is a statistical relaxation of the ordinary Poisson model [706]. It introduces a connection to linear ℓ_2-regression that we can exploit to show that a reweighted sample of size $O(\varepsilon^{-2} d \log^2 d)$ gives a good approximation of the consistent maximum likelihood estimator in this model [501].

Our experimental evaluation [501] shows that the importance sampling scheme outperforms uniform sampling for the normal GLMs. For the Poisson GLMs the result is not as remarkable and the log-likelihood approximation seems worse for large sample sizes at first glance. But as the subsample size drops below 20 %, our method captures more structure of the data. A remarkable, yet non-intuitive feature is that the approximation is capable of making better predictions than the optimal model [501]. Similar effects have been observed independently in the general setting of randomized linear algebra algorithms [461] and was attributed to an implicit regularization effect, since the distortion induced by the approximation prevents the model from overfitting the original data.

3.2.2.2 Sketches and Coresets for Bayesian Regression

Let us now focus on theoretical aspects of data compression for Bayesian regression. We point the interested reader to Section 2.4 in Volume 3 of this book series for more methodological results and applications. Bayesian regression does not assume a fixed *optimal* solution for a dataset as is required in the frequentist case. Instead, it introduces a distribution over the parameter space. The *likelihood* function $\mathcal{L}(Y|X, \beta)$ models the information that comes from the data. The *prior* distribution $p_{\text{pre}}(\beta)$ models problem-specific prior knowledge. Our goal is now to explore and characterize the *posterior* distribution $p_{\text{post}}(\beta)$, which, as a consequence of Bayes' theorem, is a compromise between the information from observed data and from the prior knowledge that we assume for the parameters[3]

$$p_{\text{post}}(\beta|X, Y) \propto \mathcal{L}(Y|X, \beta) \cdot p_{\text{pre}}(\beta).$$

Random Projections for Bayesian Regression Our work on random projections for Bayesian regression [245] extends previous work on frequentist ℓ_2-regression [145] to the Bayesian setting. Certain types of random projections studied in theoretical computer science form a so-called ε-subspace embedding. Those are linear sketches for ℓ_2-spaces,

3 Here, $a \propto b$ means that there exists a constant $c > 0$ such that $a = cb$.

which preserve the ℓ_2-norm of all vectors in a linear subspace with little distortion. The guarantee we obtain is that there exists a distribution over sketching matrices Π with a reduced target dimension $O(d/\varepsilon)$ such that

$$\forall \beta \in \mathbb{R}^d : (1 - \sqrt{\varepsilon})\|X\beta\|_2 \leq \|\Pi X\beta\|_2 \leq (1 + \sqrt{\varepsilon})\|X\beta\|_2$$

holds with high probability over the random choice of Π. This implies that it preserves the ℓ_2-regression error up to a factor of $(1 + \varepsilon)$ [145], i.e., if we solve the compressed regression problem to obtain $\tilde{\beta} \in \operatorname{argmin}_{\beta \in \mathbb{R}^d} \|\Pi(X\beta - Y)\|$ then $\tilde{\beta}$ satisfies

$$\|X\tilde{\beta} - Y\|_2 \leq (1 + \varepsilon) \min_{\beta \in \mathbb{R}^d} \|X\beta - Y\|_2.$$

For Bayesian regression we also apply an ε-subspace embedding Π to compress the data matrix $[X, Y] \in \mathbb{R}^{n \times (d+1)}$ to a sketch $[\Pi X, \Pi Y] \in \mathbb{R}^{k \times (d+1)}$ for slightly larger $k \in O(\text{poly}(d)/\varepsilon^2)$, whose dimensions notably do not depend on n. Our main finding is that the results of a *Bayesian* analysis on the sketch and on the original dataset are also similar up to little distortion, depending on the approximation parameter ε. More specifically, if we denote by $p = p_{\text{post}}(\beta|X, Y)$ and $q = p_{\text{post}}(\beta|\Pi X, \Pi Y)$ the posterior distribution on the original data and on the sketch respectively, then $p \approx_\varepsilon q$, i.e., they are close to each other. We can quantify the approximation via the Wasserstein distance [245]. This choice is especially appealing, because it relates the distance of probability measures to properties in the ℓ_2-space over which they are defined. For normal distributions this entails that their location parameters as well as their covariances are close to the original.

The aforementioned results were restricted to the most prominent case of Bayesian linear regression, namely to the basic case of a likelihood based on Gaussian distributions and a multivariate normal distribution as a prior. The model class of the prior includes the degenerate, but common non-informative choice of a uniform distribution over $\mathbb{R}^d$.

Hierarchical Models Hierarchical regression models offer an extension of the previous result to a broader class of prior distributions [247]. They present a modern statistical approach that is especially useful when information on different levels is present, e.g., in a meta-analysis, where raw data is available for some studies, but only averages for the others [699]. A hierarchical model is given by

$$p_{\text{post}}(\beta, \theta|X, Y) \propto \mathcal{L}(Y|X, \beta) \cdot p_{\text{pre}}(\beta|\theta) \cdot p_{\text{hyper}}(\theta),$$

where the prior on β on the first level depends on a *hyperparameter* θ that is again modeled via a *hyper-prior* $p_{\text{hyper}}(\theta)$ on the second level of the hierarchy. Such models can be naturally extended to model arbitrary, deep or broad hierarchies, and to model numerous different populations.

Generalized Normal Prior Distributions Generalized normal priors are another modern statistical extension that we study [247, 511]. They result from generalizing the inducing norm from ℓ_2 to ℓ_p for $p \in [1, \infty)$. Their probability density function is given by

$$f(x) = \frac{p}{2\varsigma\Gamma(1/p)} \exp\left(-\frac{|x-\mu|^p}{\varsigma^p}\right),$$

where μ is a location parameter and ς is a scale parameter. The parameter p determines the shape and heaviness of the tails. Special cases include the normal distribution for $p = 2$, the Laplace distribution for $p = 1$, and the uniform distribution on $[\mu - \varsigma, \mu + \varsigma]$ for $p \to \infty$. Generalized normal distributions have been suggested and employed as a robust alternative to model deviations from normality [438] and to model a Bayesian analogue of LASSO regression [554]. Alternatively, they can also be employed to model a higher sensitivity to outliers [513]. This is also important in the context of correcting statistical models [518].

Generalized Normal Likelihood Distributions Generalized normal likelihoods can also be treated with subspace embeddings. We note that the first such generalization for ℓ_1 was developed in the CRC 876 [637]. The case $p \in [1, \infty)$ can be approximated in a similar way as in the case of normal distributions via further generalized subspace embedding techniques for ℓ_p [709]. However, this is technically more challenging [511]. One complication is that the embedding sizes are much larger for $p > 2$ than for $p \leq 2$. The other problem is that the distortion is as large as $O((d \log d)^{1/p})$ rather than $(1 \pm \varepsilon)$. We thus use the random projection only in a preprocessing step [511] to obtain a so-called *well-conditioned basis*, which can be thought of as an ℓ_p-analogue to an orthonormal basis for ℓ_2. From this we can derive sampling probabilities such that by taking $O(d^{2p+3} \log^2 d \log(1/\varepsilon)\varepsilon^{-2})$ reweighted random samples, we achieve the desired $(1 \pm \varepsilon)$ distortion. This is in line with [709] for $p > 2$ but is slightly weaker for $p \in [1, 2]$. However, our simpler unified algorithm applies universally to both cases. Similar methods were recently developed for obtaining coresets for the p-generalized probit model [513] and are currently being extended to the Bayesian setting.

3.2.2.3 Bayesian Optimization in Embedded Subspaces

Bayesian optimization (BO) has emerged as a powerful technique for the global optimization of black-box functions that are expensive to evaluate [80, 235, 624]. Here 'black-box' means that we may evaluate an unknown but fixed objective function f at any point to observe its value, possibly with noise but without derivative information. The goal is to find

$$x^\star \in \operatorname{argmin}_{x \in \mathcal{C}} f(x)$$

over a set $\mathcal{C}$, the domain of optimization, which can represent constraints, such as a box-constraint $\mathcal{C} = [-1, 1]^D$ on a large D-dimensional domain, for instance.

The advantages of Bayesian optimization are sample efficiency, provable convergence to a global optimum, and a low computational overhead. A critical limitation is the number of parameters that BO can optimize over. This is especially true for the most common form of BO that uses Gaussian Process (GP) regression as a surrogate model for the objective function. Thus, it is not surprising that expanding BO to higher-dimensional search spaces is widely acknowledged as one of the most important goals in the field [235]. Our work [512] advances the field, both, in the theory of high-dimensional Bayesian optimization and in improving practical performance.

The idea of Bayesian optimization is to learn a Gaussian process surrogate model on the previous evaluations in order to gain knowledge on where to evaluate next by a simpler optimization of an *acquisition criterion*, e.g., the Expected Improvement (EI). Under the assumption that the objective function depends essentially only on a low d_e-dimensional *effective* subspace of an ambient high-dimensional space, we used a sparse subspace embedding matrix to perform the optimization in an intermediate subspace of dimension $O(d_e^2/\varepsilon^2)$. This solved several open problems in the area [512]:

1. It fixed the problem of large dilations that caused previous Gaussian embedding matrices to project the evaluation points out of the feasible region of optimization.
2. We provided a rigorous proof that the underlying Gaussian process is well approximated in terms of its mean and variance functions, which indicates that the sample efficiency is preserved.
3. We extended the result under mild assumptions to several highly non-linear kernel spaces, which may be of independent interest.
4. It is computationally much faster than previous and contemporary approaches due to the sparse embedding.
5. It performs among the best algorithms in practice even when the low-dimensional assumption is not satisfied, cf. [201].

We refer to Section 2.5 in Volume 3 for more research and applications using Bayesian optimization.

3.2.3 Conclusion

We introduced the concepts of coresets and sketching, which are methods for summarizing data in such a way that the reduced dataset retains provable approximation guarantees for a given computational or statistical learning problem. This enables analyzing data in resource-constrained environments such as data streams and distributed systems, sensor networks etc., which are common in embedded systems and cyber-physical systems. By reducing the data before their aggregation or analysis, our methods help to save computation time and memory requirements and support communication awareness. Consequently, this also saves resources on a lower technical level, for instance energy and bandwidth.

Originating from the theory of computing community in the early twenty-first century, those methods have paved their way into the machine learning and statistical communities over the last decade ever since the *Big Data* hype. From there, they are anticipated to spread into all kinds of technical and application domains in the near future. This also underlines the importance of integrating them into contemporary undergraduate and graduate training programs.

Research on data reduction techniques, such as coresets and sketching, is an evergrowing field from theoretical and from applied perspectives. The limitations and possibilities for relatively simple but important base problems like linear regression are now well-understood. But it remains open and challenging in many cases to extend research to more sophisticated and computationally more demanding methods such as Bayesian statistics, and neural networks.

We anticipate great advances in the field of Bayesian statistics. The advantages of those methods lie in their theoretical statistical foundation, the interpretability of their models, and their built-in quantification of uncertainty. However, normally Bayesian methods require horrendous amounts of resources. Our fundamental research has shown initial approaches for making those methods scalable and resource-efficient, and leaving still a lot of potential for future research.