Deletion-Robust Submodular Maximization: Data Summarization with "the Right to be Forgotten"

Baharan Mirzasoleiman [1] **Amin Karbasi** [2] **Andreas Krause** [1]

Abstract

How can we summarize a *dynamic* data stream when elements selected for the summary can be deleted at any time? This is an important challenge in online services, where the users generating the data may decide to exercise their right to restrict the service provider from using (part of) their data due to privacy concerns. Motivated by this challenge, we introduce the *dynamic deletion-robust submodular maximization* problem. We develop the first resilient streaming algorithm, called ROBUST-STREAMING, with a constant factor approximation guarantee to the optimum solution. We evaluate the effectiveness of our approach on several real-world applications, including summarizing (1) streams of geo-coordinates (2); streams of images; and (3) click-stream log data, consisting of 45 million feature vectors from a news recommendation task.

1. Introduction

Streams of data of massive and increasing volume are generated every second, and demand fast analysis and efficient storage, including massive clickstreams, stock market data, image and video streams, sensor data for environmental or health monitoring, to name a few. To make efficient and reliable decisions we usually need to react in real-time to the data. However, big and fast data makes it difficult to store, analyze, or make predictions. Therefore, *data summarization* – mining and extracting useful information from large data sets – has become a central topic in machine learning and information retrieval.

A recent body of research on data summarization relies on utility/scoring functions that are *submodular*. Intuitively, submodularity (Krause & Golovin, 2013) states that selecting any given data point earlier helps more than selecting it later. Hence, submodular functions can score both diversity and representativeness of a subset w.r.t. the entire dataset. Thus, many problems in data summarization require maximizing submodular set functions subject to cardinality (or more complicated hereditary constraints). Numerous examples include exemplar-based clustering (Dueck & Frey, 2007), document (Lin & Bilmes, 2011) and corpus summarization (Sipos et al., 2012), recommender systems (El-Arini & Guestrin, 2011), search result diversification (Rakesh Agrawal, 2009), data subset selection (Wei et al., 2015), and social networks analysis (Kempe et al., 2003).

Classical methods, such as the celebrated greedy algorithm (Nemhauser et al., 1978) or its accelerated versions (Mirzasoleiman et al., 2015; Badanidiyuru & Vondrák, 2014) require random access to the entire data, make multiple passes, and select elements sequentially in order to produce near optimal solutions. Naturally, such solutions cannot scale to large instances. The limitations of centralized methods inspired the design of streaming algorithms that are able to gain insights from data as it is being collected (Badanidiyuru et al., 2014; Chakrabarti & Kale, 2014; Chekuri et al., 2015; Mirzasoleiman et al., 2017).

While extracting useful information from big data in real-time promises many benefits, the development of more sophisticated methods for extracting, analyzing and using personal information has made privacy a major public issue. Various web services rely on the collection and combination of data about individuals from a wide variety of sources. At the same time, the ability to control the information an individual can reveal about herself in online applications has become a growing concern.

The *"right to be forgotten"* (with a specific mandate for protection in the European Data Protection Regulation (2012), and concrete guidelines released in 2014) allows individuals to claim the ownership of their personal information and gives them the authority to their online activities (videos, photos, tweets, etc). As an example, consider a road traffic information system that monitors traffic speeds, travel times and incidents in real time. It combines the massive amount of control messages available at the cellular network with their geo-coordinates in order to gen-

[1]ETH Zurich, Switzerland [2]Yale University, New Haven, USA. Correspondence to: Baharan Mirzasoleiman <baharanm@inf.ethz.ch>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017.

erate the area-wide traffic information service. Some consumers, while using the service and providing data, may not be willing to share information about specific locations in order to protect their own privacy. With the right to be forgotten, an individual can have certain data deleted from online database records so that third parties (e.g., search engines) can no longer trace them (Weber, 2011). Note that the data could be in many forms, including a) user's posts to an online social media, b) visual data shared by wearable cameras (e.g., Google Glass), c) behavioral patterns or feedback obtained from clicking on advertisement or news.

In this paper, we propose the first framework that offers instantaneous data summarization while preserving the right of an individual to be forgotten. We cast this problem as an instance of *robust* streaming submodular maximization where the goal is to produce a concise real-time summary in the face of data deletion requested by users. We develop ROBUST-STREAMING, a method that for a generic streaming algorithm STREAMINGALG with approximation guarantee α, ROBUST-STREAMING outputs a robust solution, against *any* m deletions from the summary at *any* given time, while preserving the *same* approximation guarantee. To the best of our knowledge, ROBUST-STREAMING is the first algorithm with such strong theoretical guarantees. Our experimental results also demonstrate the effectiveness of ROBUST-STREAMING on several practical applications.

2. Background and Related Work

Several streaming algorithms for submodular maximization have been recently developed. For monotone functions, Gomes & Krause (2010) first developed a multi-pass algorithm with $1/2-\epsilon$ approximation guarantee subject to a cardinality constraint k, using $O(k)$ memory, under strong assumptions on the way the data is generated. Later, Badanidiyuru et al. (2014) proposed the first single pass streaming algorithm with $1/2-\epsilon$ approximation under a cardinality constraint. They made no assumptions on the order of receiving data points, and only require $O(k \log k/\epsilon)$ memory. Following the same line of inquiry, Chakrabarti & Kale (2014) developed a single pass algorithm with $1/4p$ approximation guarantee for handling more general constraints such as intersections of p matroids. The required memory is unbounded and increases polylogarithmically with the size of the data. For general submodular functions, Chekuri et al. (2015) presented a randomized algorithm subject to a broader range of constraints, namely p-matchoids. Their method gives a $(2-o(1))/(8+e)p$ approximation using $O(k \log k/\epsilon^2)$ memory (k is the size of the largest feasible solution). Very recently, Mirzasoleiman et al. (2017) introduced a $(4p-1)/4p(8p+2d-1)$-approximation algorithm under a p-system and d knapsack constraints, using $O(pk \log^2(k)/\epsilon^2)$ memory.

An important requirement, which frequently arises in practice, is robustness. Krause et al. (2008) proposed the problem of robust submodular observation selection, where we want to solve $\max_{|A|\leq k} \min_{i\in[\ell]} f_i(A)$, for normalized monotonic f_i. Submodular maximization of f robust against m deletions can be cast as an instance of the above problem: $\max_{|A|\leq k} \min_{|B|\leq m} f(A\backslash B)$. The running time, however, will be exponential in m. Recently, Orlin et al. (2016) developed an algorithm with an asymptotic guarantee 0.285 for deletion-robust submodular maximization under up to $m = o(\sqrt{k})$ deletions. The results can be improved for only 1 or 2 deletions.

The aforementioned approaches aim to construct solutions that are robust against deletions in a batch mode way, without being able to update the solution set after each deletion. To the best of our knowledge, this is the first to address the general deletion-robust submodular maximization problem in the streaming setting. We also highlight the fact that our method does not require m, the number of deletions, to be bounded by k, the size of the largest feasible solution.

Very recently, submodular optimization over sliding windows has been considered, where we want to maintain a solution that considers only the last W items (Epasto et al., 2017; Jiecao et al., 2017). This is in contrast to our setting, where the guarantee is with respect to all the elements received from the stream, except those that have been deleted. The sliding window model can be easily incorporated into our solution to get a robust sliding window streaming algorithm with the possibility of m deletions in the window.

3. Deletion-Robust Model

We review the *static* submodular data summarization problem. We then formalize a novel *dynamic* variant, and constraints on time and memory that algorithms need to obey.

3.1. Static Submodular Data Summarization

In *static* data summarization, we have a large but fixed dataset V of size n, and we are interested in finding a summary that best represents the data. The representativeness of a subset is defined based on a utility function $f\colon 2^V \to \mathbb{R}_+$ where for any $A \subset V$ the function $f(A)$ quantifies how well A represents V. We define the marginal gain of adding an element $e \in V$ to a summary $A \subset V$ by $\Delta(e|A) = f(A \cup \{e\}) - f(A)$. In many data summarization applications, the utility function f satisfies *submodularity*, i.e., for all $A \subseteq B \subseteq V$ and $e \in V \setminus B$,

$$\Delta(e|A) \geq \Delta(e|B).$$

Many data summarization applications can be cast as an instance of a constrained submodular maximization:

$$\texttt{OPT} = \max_{A\in\mathcal{I}} f(A), \tag{1}$$

where $\mathcal{I} \subset 2^V$ is a given family of feasible solutions. We will denote by A^* the optimal solution, i.e. $A^* = \arg\max_{A \in \mathcal{I}} f(A)$. A common type of constraint is a cardinality constraint, i.e., $\mathcal{I} = \{A \subseteq 2^V, \text{ s.t., } |A| \leq k\}$. Finding A^* even under cardinality constraint is NP-hard, for many classes of submodular functions (Feige, 1998). However, a seminal result by Nemhauser et al. (1978) states that for a non-negative and monotone submodular function a simple greedy algorithm that starts with the empty set $S_0 = \emptyset$, and at each iteration augments the solution with the element with highest marginal gain, obtains a $(1-1/e)$ approximation to the optimum solution. For small, static data, the centralized greedy algorithm or its accelerated variants produce near-optimal solutions. However, such methods fail to scale to truly large problems.

3.2. Dynamic Data: Additions and Deletions

In *dynamic* deletion-robust submodular maximization problem, the data V is generated at a fast pace and in real-time, such that at any point t in time, a subset $V_t \subseteq V$ of the data has arrived. Naturally, we assume that $V_1 \subseteq V_2 \subseteq \cdots \subseteq V_n$, with no assumption made on the *order* or the size of the datastream. Importantly, we allow data to be *deleted* dynamically as well. We use D_t to refer to data deleted by time t, where again $D_1 \subseteq D_2 \subseteq \cdots \subseteq D_n$. Without loss of generality, below we assume that at every time step t exactly one element $e_t \in V$ is either added or deleted, i.e., $|D_t \setminus D_{t-1}| + |V_t \setminus V_{t-1}| = 1$. We now seek to solve a dynamic variant of Problem (1)

$$\texttt{OPT}_t = \max_{A_t \in \mathcal{I}_t} f(A_t) \text{ s.t. } \mathcal{I}_t = \{A : A \in \mathcal{I} \wedge A \subseteq V_t \setminus D_t\}. \tag{2}$$

Note that in general a feasible solution at time t might not be a feasible solution at a later time t'. This is particularly important in practical situations where a subset of the elements D_t should be removed from the solution. We do not make any assumptions on the order or the size of the data stream V, but we assume that the *total number of deletions is limited to* m , i.e., $|D_n| \leq m$.

3.3. Dealing with Limited Time and Memory

In principle, we could solve Problem (2) by repeatedly – at every time t – solving a static Problem (1) by restricting the ground set V to $V_t \setminus D_t$. This is impractical even for moderate problem sizes. For large problems, we may not even be able to fit V_t into the main memory of the computing device (space constraints). Moreover, in real-time applications, one needs to make decisions in a timely manner while the data is continuously arriving (time constraints).

We hence focus on *streaming algorithms* which may maintain a limited memory $M_t \subset V_t \setminus D_t$, and must have an updated feasible solution $\{A_t \mid A_t \subseteq \mathcal{M}_t, A_t \in \mathcal{I}_t\}$ to output at any given time t. Ideally, the capacity of the memory $|\mathcal{M}_t|$ should not depend on t and V_t. Whenever a new element is received, the algorithm can choose 1) to insert it into its memory, provided that the memory does not exceed a pre-specified capacity bound, 2) to replace it with one or a subset of elements in the memory (in the preemptive model), or otherwise 3) the element gets discarded and cannot be used later by the algorithm. If the algorithm receives a deletion request for a subset $D_t \subset V_t$ at time t (in which case $\mathcal{I}_t$ will be updated to accommodate this request) it has to drop D_t from $\mathcal{M}_t$ in addition to updating A_t to make sure that the current solution is feasible (all subsets $A'_t \subset V_t$ that contain an element from D_t are infeasible, i.e., $A'_t \notin \mathcal{I}_t$). To account for such losses, the streaming algorithm can only use other elements maintained in its memory in order to produce a feasible candidate solution, i.e. $A_t \subseteq \mathcal{M}_t \subseteq ((V_t \setminus V_{t-1}) \cup \mathcal{M}_{t-1}) \setminus D_t$. We say that the streaming algorithm is *robust* against m deletions, if it can provide a feasible solution $A_t \in \mathcal{I}_t$ at any given time t such that $f(A_t) \geq \tau \texttt{OPT}_t$ for some constant $\tau > 0$. Later, we show how robust streaming algorithms can be obtained by carefully increasing the memory and running multiple instances of existing streaming methods simultaneously.

4. Example Applications

We now discuss three concrete applications, with their submodular objective functions f, where the size of the datasets and the nature of the problem often require a deletion-robust streaming solution.

4.1. Summarizing Click-stream and Geolocation Data

There exists a tremendous opportunity of harnessing prevalent activity logs and sensing resources. For instance, GPS traces of mobile phones can be used by road traffic information systems (such as Google traffic, TrafficSense, Navigon) to monitor travel times and incidents in real time. In another example, stream of user activity logs is recorded while users click on various parts of a webpage such as ads and news while browsing the web, or using social media. Continuously sharing all collected data is problematic for several reasons. First, memory and communication constraints may limit the amount of data that can be stored and transmitted across the network. Second, reasonable privacy concerns may prohibit continuous tracking of users.

In many such applications, the data can be described in terms of a kernel matrix K which encodes the similarity between different data elements. The goal is to select a small subset (*active set*) of elements while maintaining a certain diversity. Very often, the utility function boils down to the following monotone submodular function (Krause & Golovin, 2013) where $\alpha > 0$ and $K_{S,S}$ is the principal submatrix of K indexed by the set S.

$$f(S) = \log\det(I + \alpha K_{S,S}) \tag{3}$$

In light of privacy concerns, it is natural to consider *participatory* models that empower users to decide what portion of their data could be made available. If a user decides not to share, or to revoke information about parts of their activity, the monitoring system should be able to update the summary to comply with users' preferences. Therefore, we use ROBUST-STREAMING to identify a robust set of the k most informative data points by maximizing Eq. (3).

4.2. Summarizing Image Collections

Given a collection of images, one might be interested in finding a subset that best summarizes and represents the collection. This problem has recently been addressed via submodular maximization. More concretely, Tschiatschek et al. (2014) designed several submodular objectives $f_1, \ldots, f_l$, which quantify different characteristics that good summaries should have, e.g., being representative w.r.t. commonly reoccurring motives. Each function either captures coverage (including *facility location*, *sum-coverage*, and *truncated graph cut*, or rewards diversity (such as *clustered facility location*, and *clustered diversity*). Then, they optimize a weighted combination of such functions

$$f_w(A) = \sum_{i=1}^{l} w_i f_i(A), \tag{4}$$

where weights are non-negative, i.e., $w_i \geq 0$, and learned via a large-margin structured prediction. We use their learned mixtures of submodular functions in our image summarization experiments. Now, consider a situation where a user wants to summarize a large collection of her photos. If she decides to delete some of the selected photos in the summary, she should be able to update the result without processing the whole collection from scratch. ROBUST-STREAMING can be used as an appealing method.

5. Robust-Streaming Algorithm

In this section, we first elaborate on why naively increasing the solution size does not help. Then, we present our main algorithm, ROBUST-STREAMING, for deletion-robust streaming submodular maximization. Our approach builds on the following key ideas: 1) simultaneously constructing non-overlapping solutions, and 2) appropriately merging solutions upon deleting an element from the memory.

5.1. Increasing the Solution Size Does Not Help

One of the main challenges in designing streaming solutions is to immediately discover whether an element received from the data stream at time t is good enough to be added to the memory $\mathcal{M}_t$. This decision is usually made based on the added value or marginal gain of the new element which in turn depends on the previously chosen elements in the memory, i.e., $\mathcal{M}_{t-1}$. Now, let us consider the opposite scenario when an element e should be deleted from the memory at time t. Since now we have a smaller context, submodularity guarantees that the marginal gains of the elements added to the memory after e was added, could have only increased if e was not part of the stream (diminishing returns). Hence, if some elements had large marginal values to be included in the memory before the deletion, they still do after the deletion. Based on this intuition, a natural idea is to keep a solution of a bigger size, say $m+k$ (rather than k) for at most m deletions. However, this idea does not work as shown by the following example.

Bad Example (Coverage): Consider a collection of n subsets $V = \{B_1, \ldots, B_n\}$, where $B_i \subseteq \{1, \ldots, n\}$, and a coverage function $f(A) = |\cup_{i \in A} B_i|$, $A \subseteq V$. Suppose we receive $B_1 = \{1, \ldots, n\}$, and then $B_i = \{i\}$ for $2 \leq i \leq n$ from the stream. Streaming algorithms that select elements according to their marginal gain and are allowed to pick $k + m$ elements, will only pick up B_1 upon encounter (as other elements provide no gain), and return $A_n = \{B_1\}$ after processing the stream. Hence, if B_1 is deleted after the stream is received, these algorithms return the empty set $A_n = \emptyset$ (with $f(A_n) = 0$). An optimal algorithm which knows that element B_1 will be deleted, however, will return set $A_n = \{B_2, \ldots, B_{k+2}\}$, with value $f(A_n) = k + 1$. Hence, standard streaming algorithms fail arbitrarily badly even under a single deletion (i.e., $m = 1$), even when we allow them to pick sets larger than k.

In the following we show how we can solve the above issue by carefully constructing not one but multiple solutions.

5.2. Building Multiple Solutions

As stated earlier, the existing one-pass streaming algorithms for submodular maximization work by identifying elements with marginal gains above a carefully chosen threshold. This ensures that any element received from the stream which is fairly similar to the elements of the solution set is discarded by the algorithm. Since elements are chosen as diverse as possible, the solution may suffer dramatically in case of a deletion.

One simple idea is to try to find m (near) duplicates for each element e in the memory, i.e., find e' such that $f(e') = f(e)$ and $\Delta(e'|e) = 0$ (Orlin et al., 2016). This way if we face m deletions we can still find a good solution. The drawback is that even one duplicate may not exist in the data stream (see the bad example above), and we may not be able to recover for the deleted element. Instead, what we will do is to construct non-overlapping solutions such that once we experience a deletion, only one solution gets affected.

In order to be robust against m deletions, we run a cascading chain of r instances of STREAMINGALGs as follows. Let $\mathcal{M}_t = M_t^{(1)}, M_t^{(2)}, \ldots, M_t^{(r)}$ denote the con-

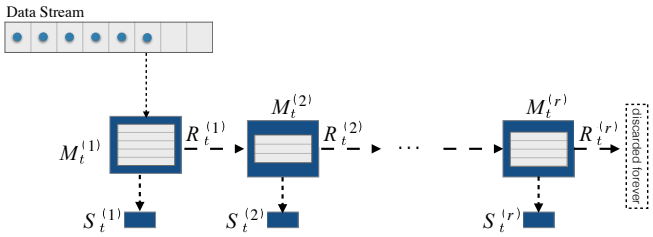


Figure 1: ROBUST-STREAMING uses r instances of a generic STREAMINGALG to construct r non-overlapping memories at any given time t, i.e., $M_t^{(1)}, M_t^{(2)}, \ldots, M_t^{(r)}$. Each instance produces a solution $S_t^{(i)}$ and the solution returned by ROBUST-STREAMING is the first valid solution $S_t = \{S_t^{(i)} | i = \min j \in [1 \cdots r], M_t^{(i)} \neq \text{null}\}$.

tent of their memories at time t. When we receive a new element $e \in V_t$ from the data stream at time t, we pass it to the first instance of STREAMINGALG$^{(1)}$. If STREAMINGALG$^{(1)}$ discards e, the discarded element is cascaded in the chain and is passed to its successive algorithm, i.e. STREAMINGALG$^{(2)}$. If e is discarded by STREAMINGALG$^{(2)}$, the cascade continues and e is passed to STREAMINGALG$^{(3)}$. This process continues until either e is accepted by one of the instances or discarded for good. Now, let us consider the case where e is accepted by the i-th instance, SIEVE-STREAMING$^{(i)}$, in the chain. As discussed in Section 3.3, STREAMINGALG may choose to discard a set of points $R_t^{(i)} \subset M_t^{(i)}$ from its memory before inserting e, i.e., $M_t^{(i)} \leftarrow M_t^{(i)} \cup \{e\} \setminus R_t^{(i)}$. Note that $R_t^{(i)}$ is empty, if e is inserted and no element is discarded from $M_t^{(i)}$. For every discarded element $r \in R_t^{(i)}$, we start a new cascade from $(i+1)$-th instance, STREAMINGALG$^{(i+1)}$.

Note that in the worst case, every element of the stream can go once through the whole chain during the execution of the algorithm, and thus the processing time for each element scales linearly by r. An important observation is that at any given time t, all the memories $M_t^{(1)}, M_t^{(2)}, \cdots, M_t^{(r)}$ contain disjoint sets of elements. Next, we show how this data structure leads to a deletion-robust streaming algorithm.

5.3. Dealing with Deletions

Equipped with the above data structure shown in Fig. 1, we now demonstrate how deletions can be treated. Assume an element e_d is being deleted from the memory of the j-th instance of STREAMINGALG$^{(j)}$ at time t, i.e., $M_t^{(j)} \leftarrow M_t^{(j)} \setminus \{e_d\}$. As discussed in Section 5.1, the solution of the streaming algorithm can suffer dramatically from a deletion, and we may not be able to restore the quality of the solution by substituting similar elements. Since there is no guarantee for the quality of the solution after a deletion, we remove STREAMINGALG$^{(j)}$ from the chain by making $R_t^{(j)} = \text{null}$ and for all the remaining elements in its memory $M_t^{(j)}$, namely, $R_t^{(j)} \leftarrow M_t^{(j)} \setminus \{e_d\}$, we start a new cascade from $j+1$-th instance, STREAMINGALG$^{(j+1)}$.

The key reason why the above algorithm works is that the guarantee provided by the streaming algorithm is independent of the order of receiving the data elements. Note that at any point in time, the first instance i of the algorithm with $M_t^{(i)} \neq$ null has processed all the elements from the stream V_t (not necessarily in the order the stream is originally received) except the ones deleted by time t, i.e., D_t. Therefore, we can guarantee that STREAMINGALG$^{(i)}$ provides us with its inherent α-approximation guarantee for reading $V_t \setminus D_t$. More precisely, $f(S_t^{(i)}) \geq \alpha \texttt{OPT}_t$, where $\texttt{OPT}_t$ is the optimum solution for the constrained optimization problem (2) when we have m deletions.

In case of adversary deletions, there will be one deletion from the solution of m instances of STREAMINGALG in the chain. Therefore, having $r = m + 1$ instances, we will remain with only one STREAMINGALG that gives us the desired result. However, as shown later in this section, if the deletions are i.i.d. (which is often the case in practice), and we have m deletions in expectation, we need r to be much smaller than $m+1$. Finally, note that we do not need to assume that $m \leq k$ where k is the size of the largest feasible solution. The above idea works for arbitrary $m \leq n$.

The pseudocode of ROBUST-STREAMING is given in Algorithm 1. It uses $r \leq m+1$ instances of STREAMINGALG as subroutines in order to produce r solutions. We denote by $S_t^{(1)}, S_t^{(1)}, \ldots, S_t^{(r)}$ the solutions of the r STREAMINGALGs at any given time t. We assume that an instance i of STREAMINGALG$^{(i)}$ receive an input element and produces a solution $S_t^{(i)}$ based on the input. It may also change its memory content $M_t^{(i)}$, and discard a set $R_t^{(i)}$. Among all the remained solutions (i.e., the ones that are not "null"), it returns the first solution in the chain, i.e. the one with the lowest index.

Theorem 1 *Let* STREAMINGALG *be a 1-pass streaming algorithm that achieves an α-approximation guarantee for the constrained maximization problem (2) with an update time of T, and a memory of size M when there is no deletion. Then* ROBUST-STREAMING *uses $r \leq m+1$ instances of* STREAMINGALGs *to produce a feasible solution $S_t \in \mathcal{I}_t$ (now $\mathcal{I}_t$ encodes deletions in addition to constraints) such that $f(S_t) = \alpha \texttt{OPT}_t$ as long as no more than m elements are deleted from the data stream. Moreover,* ROBUST-STREAMING *uses a memory of size rM, and has worst case update time of $O(r^2MT)$, and average update time of $O(rT)$.*

The proofs can be found in the appendix. In Table 1 we combine the result of Theorem 1 with the existing streaming algorithms that satisfy our requirements.

Algorithm 1 ROBUST-STREAMING

Input: data stream V_t, deletion set D_t, $r \leq m+1$.
Output: solution S_t at any time t.

$t = 1$, $M_t^{(i)} = 0$, $S_t^{(i)} = \emptyset$ $\quad \forall i \in [1 \cdots r]$
while $(\{V_t \setminus V_{t-1}\} \cup \{D_t \setminus D_{t-1}\} \neq \emptyset)$ **do**
  if $\{D_t \setminus D_{t-1}\} \neq \emptyset$ **then**
    $e_d \leftarrow \{D_t \setminus D_{t-1}\}$
    Delete(e_d)
  else
    $e_t \leftarrow \{V_t \setminus V_{t-1}\}$
    Add(1, e_t)
  end if
  $t = t + 1$
  $S_t = \{S_t^{(i)} | \ i = \min\{j \in [1 \cdots r],\ M_t^{(j)} \neq \text{null}\}\}$
end while

function Add(i, R)
  for $e \in R$ **do**
    $[R_t^{(i)}, M_t^{(i)}, S_t^{(i)}] =$STREAMINGALG$^{(i)}(e)$
    if $R_t^{(i)} \neq \emptyset$ **and** $i < r$ **then**
      Add($i + 1$, $R_t^{(i)}$)
    end if
  end for
end function

function Delete(e)
  for $i = 1$ to r **do**
    if $e \in M_t^{(i)}$ **then**
      $R_t^{(i)} = M_t^{(i)} \setminus \{e\}$
      $M_t^{(i)} \leftarrow$ null
      Add($i + 1$, $R_t^{(i)}$)
      return
    end if
  end for
end function

Theorem 2 *Assume each element of the stream is deleted with equal probability $p = m/n$, i.e., in expectation we have m deletions from the stream. Then, with probability $1 - \delta$,* ROBUST-STREAMING *provides an α-approximation as long as*

$$r \geq \left(\frac{1}{1-p}\right)^k \log\left(1/\delta\right).$$

Theorem 2 shows that for fixed k, δ and p, a *constant* number r of STREAMINGALGs is sufficient to support $m = pn$ (expected) deletions *independently* of n. In contrast, for adversarial deletions, as analyzed in Theorem 1, $pn + 1$ copies of STREAMINGALG are required, which grows linearly in n. Hence, the required dependence of r on m is much milder for random than adversarial deletions. This is also verified by our experiments in Section 6.

6. Experiments

We address the following questions: 1) How much can ROBUST-STREAMING recover and possibly improve the performance of STREAMINGALG in case of deletions? 2) How much does the time of deletions affect the performance? 3) To what extent does deleting representative vs. random data points affect the performance? To this end, we run ROBUST-STREAMING on the applications we described in Section 4, namely, image collection summarization, summarizing stream of geolocation sensor data, as well as summarizing a clickstream of size 45 million.

Throughout this section we consider the following streaming algorithms: SIEVE-STREAMING (Badanidiyuru et al., 2014), STREAM-GREEDY (Gomes & Krause, 2010), and STREAMING-GREEDY (Chekuri et al., 2015). We allow all streaming algorithms, including the non-preemptive SIEVE-STREAMING, to update their solution after each deletion. We also consider a stronger variant of SIEVE-STREAMING, called EXTSIEVE, that aims to pick $k \cdot r$ elements to protect for deletions, i.e., is allowed the same memory as ROBUST-STREAMING. After the deletions, the remaining solution is pruned to k elements.

To compare the effect of deleting representative elements to the that of deleting random elements from the stream, we use two stochastic variants of the greedy algorithm, namely, STOCHASTIC-GREEDY (Mirzasoleiman et al., 2015) and RANDOM-GREEDY (Buchbinder et al., 2014). This way we introduce randomness into the deletion process in a principled way. Hence, we have:

STOCHASTIC-GREEDY (SG): Similar to the the greedy algorithm, STOCHASTIC-GREEDY starts with an empty set and adds one element at each iteration until obtains a solution of size m. But in each step it first samples a random set R of size $(n/m)\log(1/\epsilon)$ and then adds an element from R to the solution which maximizes the marginal gain.

RANDOM-GREEDY (RG): RANDOM-GREEDY iteratively selects a random element from the top m elements with the highest marginal gains, until finds a solution of size m.

For each deletion method, the m data points are deleted either while receiving the data (where the steaming algorithms have the chance to update their solutions by selecting new elements) or after receiving the data (where there is no chance of updating the solution with new elements). Finally, *the performance of all algorithms are normalized against the utility obtained by the centralized algorithm that knows the set of deleted elements in advance.*

6.1. Image Collection Summarization

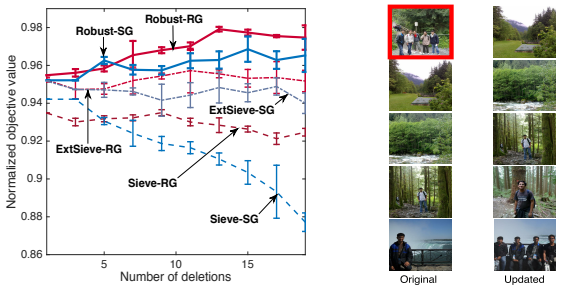
We first apply ROBUST-STREAMING to a collection of 100 images from Tschiatschek et al. (2014). We used

Algorithm	Problem	Constraint	Appr. Fact.	Memory
ROBUST + SIEVE-STREAMING [BMKK'14]	Mon. Subm.	Cardinality	$1/2-\epsilon$	$O(mk\log k/\epsilon)$
ROBUST + f-MSM [CK'14]	Mon. Subm.	p-matroids	$\frac{1}{4p}$	$O(mk(\log\lvert V\rvert)^{O(1)})$
ROBUST + STREAMING-GREEDY [CGQ'15]	Non-mon. Subm.	p-matchoid	$\frac{(1-\varepsilon)(2-o(1))}{(8+\epsilon)p}$	$O(mk\log k/\epsilon^2)$
ROBUST + STREAMING-LOCAL SEARCH [MJK'17]	Non-mon. Subm.	p-system + d knapsack	$\frac{(1-\varepsilon)(4p-1)}{4p(8p+2d-1)}$	$O(mpk\log^2 k/\epsilon^2)$

Table 1: ROBUST-STREAMING combined with 1-pass streaming algorithms can make them robust against m deletions.

the weighted combination of 594 submodular functions either capturing coverage or rewarding diversity (c.f. Section 4.2). Here, despite the small size of the dataset, computing the weighted combination of 594 functions makes the function evaluation considerably expensive.

Fig. 2a compares the performance of SIEVE-STREAMING with its robust version ROBUST-STREAMING for $r=3$ and solution size k=5. Here, we vary the number m of deletions from 1 to 20 after the whole stream is received. We see that ROBUST-STREAMING maintains its performance by updating the solution after deleting subsets of data points imposed by different deletion strategies. It can be seen that, even for a larger number m of deletions, ROBUST-STREAMING, run with parameter $r<m$, is able to return a solution competitive with the strong centralized benchmark that knows the deleted elements beforehand. For the image collection, we were not able to compare the performance of STREAM-GREEDY with its robust version due to the prohibitive running time. Fig. 2b shows an example of an updated image summary returned by ROBUST-STREAMING after deleting the first image from the summary.



(a) Images (b) Images

Figure 2: Performance of ROBUST-STREAMING vs SIEVE-STREAMING for different deletion strategies (SG, RG), at the end of stream, on a collection of 100 images. Here we fix $k=5$ and $r=3$. a) performance of ROBUST-STREAMING and SIEVE-STREAMING normalized by the utility obtained by greedy that knows the deleted elements beforehand. b) updated solution returned by ROBUST-STREAMING after deleting the first images in the summary.

6.2. Summarizing a stream of geolocation data

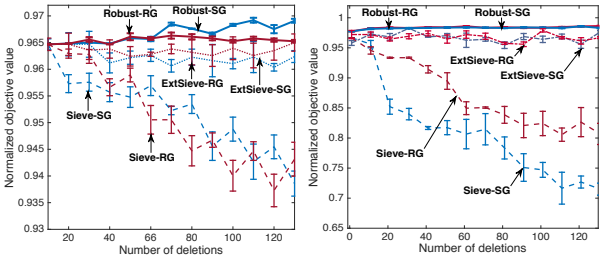
Next we apply ROBUST-STREAMING to the active set selection objective described in Section 4.1. Our dataset consists of 3,607 geolocations, collected during a one hour bike ride around Zurich (Fatio, 2015). For each pair of points i and j we used the corresponding (latitude, longitude) coordinates to calculate their distance in meters $d_{i,j}$ and chose a Gaussian kernel $K_{i,j}=\exp(-d_{i,j}^2/h^2)$ with $h=1500$.

Fig. 3e shows the dataset where red and green triangles show a summary of size 10 found by SIEVE-STREAMING, and the updated summary provided by ROBUST-STREAMING with $r=5$ after deleting $m=70\%$ of the datapoints. Fig. 3a and 3c compare the performance of SIEVE-STREAMING with its robust version when the data is deleted after or during the stream, respectively. As we see, ROBUST-STREAMING provides a solution very close to the hindsight centralized method. Fig. 3b and 3d show similar behavior for STREAM-GREEDY. Note that deleting data points via STOCHASTIC-GREEDY or RANDOM-GREEDY are much more harmful on the quality of the solution provided by STREAM-GREEDY. We repeated the same experiment by dividing the map into grids of length 2km. We then considered a partition matroid by restricting the number of points selected from each grid to be 1. The red and green triangles in Fig. 3f are the summary found by STREAMING-GREEDY and the updated summary provided by ROBUST-STREAMING after deleting the shaded area in the figure.

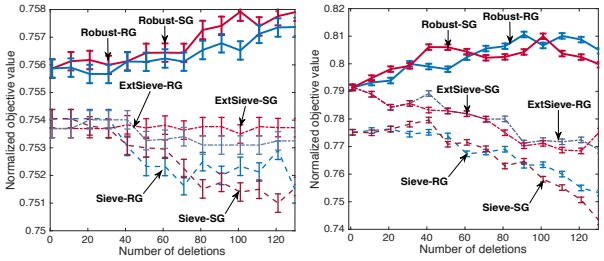
6.3. Large scale click through prediction

For our large-scale experiment we consider again the active set selection objective, described in Section 4.1. We used *Yahoo! Webscope* data set containing 45,811,883 user click logs for news articles displayed in the Featured Tab of the Today Module on Yahoo! Front Page during the first ten days in May 2009 (Yahoo, 2012). For each visit, both the user and shown articles are associated with a feature vector of dimension 6. We take their outer product, resulting in a feature vector of size 36.

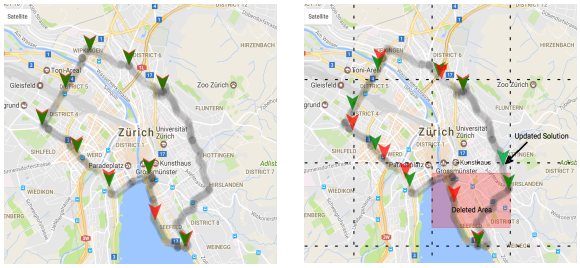
The goal was to predict the user behavior for each displayed article based on historical clicks. To do so, we considered the first 80% of the data (for the fist 8 days) as our training set, and the last 20% (for the last 2 days) as our test set. We used Vowpal-Wabbit (Langford et al., 2007) to train a linear classifier on the full training set. Since only 4% of the data points are clicked, we assign a weight of 10 to each



(a) SIEVE-STREAMING, at end (b) STREAM-GREEDY, at end



(c) SIEVE-STREAMING, during (d) STREAM-GREEDY, during

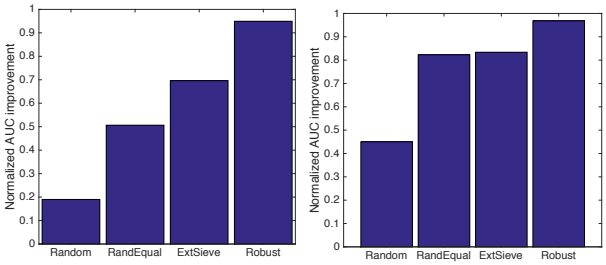


(e) Cardinality constraints (f) Matroid constraints

Figure 3: ROBUST-STREAMING vs SIEVE-STREAMING and STREAM-GREEDY for different deletion strategies (SG, RG) on geolocation data. We fix $k = 20$ and $r = 5$. a) and c) show the performance of robustified SIEVE-STREAMING, whereas b) and d) show performance for robustified STREAM-GREEDY. a) and b) consider the performance after deletions at the end of the stream, while c) and d) consider average performance while deletions happen during the stream. e) red and green triangles show a set of size 10 found by SIEVE-STREAMING and the updated solution found by ROBUST-STREAMING where 70% of the points are deleted. f) set found by STREAMING-GREEDY, constrained to pick at most 1 point per grid cell (matroid constraint). Here $r = 5$, and we deleted the shaded area.

clicked vector. The AUC score of the trained classifier on the test set was 65%. We then used ROBUST-STREAMING and SIEVE-STREAMING to find a representative subset of size k consisting of $k/2$ clicked and $k/2$ not-clicked examples from the training data. Due to the massive size of the dataset, we used Spark on a cluster of 15 quad-core machines with 32GB of memory each. We partitioned the training data to the machines keeping its original order. We ran ROBUST-STREAMING on each machine to find a summary of size $k/15$, and merged the results to obtain the final summary of size k. We then start deleting the data uniformly at random until we left with only 1% of the data, and trained another classifier on the remaining elements from the summary.

Fig. 4a compares the performance of ROBUST-STREAMING for a fixed active set of size $k = 10,000$, and $r = 2$ with random selection, randomly selecting equal numbers of clicked and not-clicked vectors, and using SIEVE-STREAMING for selecting equal numbers of clicked and not-clicked data points. The y-axis shows the improvement in AUC score of the classifier trained on a summary obtained by different algorithms over random guessing (AUC=0.5), normalized by the AUC score of the classifier trained on the whole training data. To maximize fairness, we let other baselines select a subset of $r.k$ elements before deletions. Fig. 4b shows the same quantity for $r = 5$. It can be seen that a slight increase in the amount of memory helps boosting the performance for all the algorithms. However, ROBUST-STREAMING benefits from the additional memory the most, and can almost recover the performance of the classifier trained on the *full training data, even after 99% deletion*.



(a) Yahoo! Webscope, $r = 2$ (b) Yahoo! Webscope, $r = 5$

Figure 4: ROBUST-STREAMING vs random unbalanced and balanced selection and SIEVE-STREAMING selecting equal numbers of clicked and not-clicked data points, on 45,811,883 feature vectors from *Yahoo! Webscope* data. We fix $k = 10,000$ and delete 99% of the data points.

7. Conclusion

We have developed the first deletion-robust streaming algorithm – ROBUST-STREAMING – for constrained submodular maximization. Given any single-pass streaming algorithm STREAMINGALG with α-approximation guarantee, ROBUST-STREAMING outputs a solution that is robust against m deletions. The returned solution also satisfies an α-approximation guarantee w.r.t. to the solution of the optimum centralized algorithm that knows the set of m deletions in advance. We have demonstrated the effectiveness of our approach through an extensive set of experiments.

Acknowledgements

This research was supported by ERC StG 307036, a Microsoft Faculty Fellowship, DARPA Young Faculty Award (D16AP00046), Simons-Berkeley fellowship and an ETH Fellowship. This work was done in part while Amin Karbasi, and Andreas Krause were visiting the Simons Institute for the Theory of Computing.

References

Babaei, Mahmoudreza, Mirzasoleiman, Baharan, Jalili, Mahdi, and Safari, Mohammad Ali. Revenue maximization in social networks through discounting. *Social Network Analysis and Mining*, 3(4):1249–1262, 2013.

Badanidiyuru, Ashwinkumar and Vondrák, Jan. Fast algorithms for maximizing submodular functions. In *SODA*, 2014.

Badanidiyuru, Ashwinkumar, Mirzasoleiman, Baharan, Karbasi, Amin, and Krause, Andreas. Streaming submodular maximization: Massive data summarization on the fly. In *KDD*, 2014.

Buchbinder, Niv, Feldman, Moran, Naor, Joseph Seffi, and Schwartz, Roy. Submodular maximization with cardinality constraints. In *SODA*, 2014.

Chakrabarti, Amit and Kale, Sagar. Submodular maximization meets streaming: Matchings, matroids, and more. *IPCO*, 2014.

Chekuri, Chandra, Gupta, Shalmoli, and Quanrud, Kent. Streaming algorithms for submodular function maximization. In *ICALP*, 2015.

Dueck, Delbert and Frey, Brendan J. Non-metric affinity propagation for unsupervised image categorization. In *ICCV*, 2007.

El-Arini, Khalid and Guestrin, Carlos. Beyond keyword search: Discovering relevant scientifiditerature. In *KDD*, 2011.

Epasto, Alessandro, Lattanzi, Silvio, Vassilvitskii, Sergei, and Zadimoghaddam, Morteza. Submodular optimization over sliding windows. In *WWW*, 2017.

Fatio, Philipe. https://refind.com/fphilipe/topics/open-data, 2015.

Feige, Uriel. A threshold of ln n for approximating set cover. *Journal of the ACM*, 1998.

Gomes, Ryan and Krause, Andreas. Budgeted nonparametric learning from data streams. In *ICML*, 2010.

guidelines. http://data.consilium.europa.eu/doc/document/ST-9565-2015-INIT/en/pdf.

Jiecao, Chen, Nguyen, Huy L., and Zhang, Qin. Submodular optimization over sliding windows. 2017. preprint, https://arxiv.org/abs/1611.00129.

Kempe, David, Kleinberg, Jon, and Tardos, Éva. Maximizing the spread of influence through a social network. In *KDD*, 2003.

Krause, Andreas and Golovin, Daniel. Submodular function maximization. In *Tractability: Practical Approaches to Hard Problems*. Cambridge University Press, 2013.

Krause, Andreas, McMahon, H Brendan, Guestrin, Carlos, and Gupta, Anupam. Robust submodular observation selection. *Journal of Machine Learning Research*, 2008.

Langford, John, Li, Lihong, and Strehl, Alex. Vowpal wabbit online learning project, 2007.

Lin, Hui and Bilmes, Jeff. A class of submodular functions for document summarization. In *NAACL/HLT*, 2011.

Mirzasoleiman, Baharan, Badanidiyuru, Ashwinkumar, Karbasi, Amin, Vondrak, Jan, and Krause, Andreas. Lazier than lazy greedy. In *AAAI*, 2015.

Mirzasoleiman, Baharan, Jegelka, Stefanie, and Krause, Andreas. Streaming non-monotone submodular maximization: Personalized video summarization on the fly. 2017. preprint, https://arxiv.org/abs/1706.03583.

Nemhauser, George L., Wolsey, Laurence A., and Fisher, Marshall L. An analysis of approximations for maximizing submodular set functions - I. *Mathematical Programming*, 1978.

Orlin, James B, Schulz, Andreas S, and Udwani, Rajan. Robust monotone submodular function maximization. *IPCO*, 2016.

Rakesh Agrawal, Sreenivas Gollapudi, Alan Halverson Samuel Ieong. Diversifying search results. In *WSDM*, 2009.

Regulation, European Data Protection. http://ec.europa.eu/justice/data-protection/document/review2012/com_2012_11_en.pdf, 2012.

Sipos, Ruben, Swaminathan, Adith, Shivaswamy, Pannaga, and Joachims, Thorsten. Temporal corpus summarization using submodular word coverage. In *CIKM*, 2012.

Tschiatschek, Sebastian, Iyer, Rishabh K, Wei, Haochen, and Bilmes, Jeff A. Learning mixtures of submodular functions for image collection summarization. In *NIPS*, 2014.

Weber, R. The right to be forgotten: More than a pandora's box? https://www.jipitec.eu/issues/jipitec-2-2-2011/3084, 2011.

Wei, Kai, Iyer, Rishabh, and Bilmes, Jeff. Submodularity in data subset selection and active learning. In *ICML*, 2015.

Yahoo. Yahoo! academic relations. r6a, yahoo! front page today module user click log dataset, version 1.0, 2012. URL http://Webscope.sandbox.yahoo.com.

Supplementary Materials.

Proof of Theorem 1

Approximation guarantee. First, we proof that ROBUST-STREAMING provides an α-approximation guarantee, using $O(r.M)$ memory. Note that since we experience at most m adversarial deletions, for $r = m + 1$, one of the $(m + 1)$ instances of STREAMINGALG has never experienced any deletion. Hence, there is at least one instance left. We show that among all instances left (i.e., did not face any deletions from their memory), the one with the lowest index provides the claimed approximation guarantee. Let us assume that at time t the instance i of STREAMINGALG is the remaining one with lowest index. As a result, the content of all $M_t^{(1)}$, $M_t^{(2)}$, ..., $M_t^{(i-1)}$ memories (in some order) has been passed to $M_t^{(i)}$. This implies that $M_t^{(i)}$ has seen all the elements in the data stream V_t except the ones deleted D_t by time t. Since STREAMINGALG provides an α approximation for reading any sequence (*independent of the order* of reading its elements), the instance i of STREAMINGALG provides an α approximation for reading $V_t \setminus D_t$. Hence $f(S_t^{(i)}) \geq \alpha \texttt{OPT}_t$ where $\texttt{OPT}_t$ is the optimum solution for the constrained optimization problem (2) when we have m deletions. Clearly, in order to run ROBUST-STREAMING we need $O(r \cdot M)$ memory as we have at most r instances of STREAMINGALG running simultaneously, each requiring memory of size M.

Update time. We now calculate an upper bound on the update time of the algorithm. The new element received from the stream is first processed by STREAMINGALG$^{(1)}$. If it is accepted, it may result in discarding at most M elements from the memory $M_t^{(1)}$ of STREAMINGALG$^{(1)}$. In the worst case, the discarded elements from $M_t^{(1)}$ result in discarding all the M elements from the memory $M_t^{(2)}$ of STREAMINGALG$^{(2)}$, which in turn may result in discarding all the M elements from the memory $M_t^{(3)}$ of STREAMINGALG$^{(3)}$. This process continues until all the elements int the memory of all the STREAMINGALGs are discarded. Taking a sum over the number of discarded elements, the update time will be scaled by

$$Mr + M(r-1) + M(r-2) + M = O(M.r^2) \tag{5}$$

The update time in case of a deletion can be calculated in a similar manner. For calculating the average update time, we note that in the worst case, every element can go through the entire chain of r instances of STREAMINGALGs. Therefore, if all the elements gets discarded by all the instances of STREAMINGALG at some point during the run of ROBUST-STREAMING, we have a total running time of $O(n.r.T)$, where n is the size of the stream. Hence, the average update time per element is $O(r.T)$.

Proof of Theorem 2

If each element has probability p of being deleted, the probability that no elements is deleted from the solution set of at least one of the streaming algorithms is

$$1 - \Big(1 - \big(1-p\big)^k\Big)^r$$

and we want this probability to be greater than $1 - \delta$. Hence, we have

$$\begin{aligned}
&1 - \Big(1 - \big(1-p\big)^k\Big)^r \geq 1 - \delta \\
&r \log \Big(1 - \big(1-p\big)^k\Big)^{-1} \geq \log(1/\delta) \\
&r(1-p)^k \geq \log(1/\delta) \\
&r \geq (1-p)^{-k} \log(1/\delta)
\end{aligned}$$

Having $p = \frac{\alpha}{n} \leq 1$, r is inversly proportional to $(1 - \frac{\alpha}{n})^k$. Therefore, for fixed k, δ and p, a *constant* number r is sufficient to support $m = pn$ *independently* of n.