Data Subset Selection via Submodularity: Offline and Streaming Settings

Fan Jue (A0221578B), Tian Xiao (A0220592L)

Abstract

Data subset selection is concerned with selecting a high-quality subset of limited size from a huge data pool such that data curators can use the subset to train a good machine learning model even if they have limited budget and resources. This objective aligns with the goal of submodular maximization (under cardinality constraints), which is to find the subset of highest utility when the function mapping each subset to its utility is submodular. Notably, submodular maximization has an efficient approximation algorithm via the greedy approach, thus we discuss how to formulate the utility function such that the efficient solution can be applied to data subset selection. We also consider a special setting where the selected data may be deleted and thus the utility function needs to be deletion-robust to maintain high utility after data deletions. Secondly, as data in real-life may arrive in the form of a stream, we discuss submodular maximization algorithms in the streaming setting (and their application to data subset selection). We further explore deletion-robust variants of these algorithms in the streaming setting.

1 Introduction

Submodular functions [19] are a well-studied class of set functions that have many desirable properties. For example, when the set function is monotone submodular and the goal is to select a subset of limited size with as high value as possible, even a **simple greedy algorithm** that always selects the datum that increases the function value most performs reasonably well, which reduces the need to expensively iterate through all feasible subsets. Therefore, submodular functions have a wide range of use cases in data-intensive domains such as *machine learning* (ML), which studies how a *model* can learn the underlying relationships or patterns from a dataset.

Data subset selection (SS) [12] is a sub-field of ML that studies how to select the best subset S of data from the data pool D, because it is prohibitively expensive for data curators to acquire and use the entire data pool D in terms of financial cost, resources and storage. One intuitive approach is **to define a submodular utility function over subsets of** D, so that the data curator can exploit efficient algorithms, such as the simple greedy algorithm described above, to select a high-quality subset. Following this idea, a variety of submodular utility functions have been devised based on different ML models. As a special use case, laws such as *General Data Protection Regulations* (GDPR) have entitled the *right to be forgotten* [3], which allows data owners to delete their data from the data pool D or selected subset S. Thus, it is important for the utility function to be *deletion-robust*, which ensures that the utility of S remains high even after deletions.

Nonetheless, even with the simple greedy algorithm, data curators must still access the entire data pool D to identify the best datum at each iteration, which requires storage of the full dataset D and can thus

be costly. Hence, they may instead prefer to process the data incrementally. Moreover, real-life data may not be nicely compiled together and more than often they arrive in the fashion of a *stream* [5] (i.e., data arrive sequentially over time). For example, in time-sensitive use cases such as stock trades [24] or social media [23], it is impractical to wait for all data to arrive before making a selection. In such cases, the greedy algorithm is not applicable because there is no access to the entire data pool D at each iteration. Therefore, it becomes important **how submodularity can still be leveraged to achieve efficiency in the *streaming* setting**, in constrast to the *offline* setting discussed earlier.

In this report, we will discuss various strategies of SS based on submodularity in both offline and streaming settings. The structure of this report is as follows:

- In Sec. 2, we introduce the basics about submodularity and traditional submodular maximization algorithms. We also provide a brief review of SS and present the notations used throughout the report;
- In Sec. 3, we explain how submodularity and its algorithms can be applied to SS in the offline setting, where we focus on the design of submodular utility functions in order to select high-quality subsets. Additionally, we discuss deletion-robust variants of these submodular functions which select subsets that maintain high utility even after data deletions;
- In Sec. 4, we turn to the streaming setting where traditional submodular maximization algorithms do not apply. We present streaming submodular maximization algorithms and their deletion-robust variants, which can be used in conjunction with the submodular functions discussed in Sec. 3.

2 Backgrounds

2.1 Submodularity

Let D be a dataset with n data and $f : \mathcal{P}(D) \to \mathbb{R}$ ($\mathcal{P}(\cdot)$ denotes power set) be a *utility function* mapping each subset of D to a real-valued utility. The function f is *monotone* if adding elements to any subset does not decrease its utility, that is, $\forall S \subseteq T \subseteq D\ [f(S) \leq f(T)]$. For any subset $S \subset D$, we further denote the *marginal gain* an element $d \in D \setminus S$ brings to S as $\Delta_f(d|S) \coloneqq f(S \cup \{d\}) - f(S)$. The function f is *submodular* if adding elements to a smaller subset yields a marginal gain that is at least as large as adding to a larger subset, that is, $\forall S \subseteq T \subseteq D\ \forall j \in D \setminus T\ [\Delta_f(d|S) \geq \Delta_f(d|T)]$.

In *cardinality-constrained submodular maximization*, one aims to find the set S_{OPT} with the highest possible utility among all subsets of size at most k (i.e., $\max_{S \subseteq D, |S| \leq k} f(S)$). A bruteforce search over all such subsets takes $O\left(n^k\right)$ time and is computationally intractable. Fortunately, when the utility function f is monotone and submodular, a well-known simple yet effective approach is the SimpleGreedy algorithm [17], which always selects the element that provides the highest marginal gain to the currently selected subset for k iterations. The SimpleGreedy algorithm runs in only $O(kn)$ time and is guaranteed to return a subset S^* with utility at least $(1-1/e) \cdot f(S_{\text{OPT}})$. As a further improvement in runtime, the LazyGreedy algorithm [13] maintains a priority queue of marginal gains. At each iteration, only a small number of top marginal gains are recomputed. The priority queue ensures that a datum's marginal gain is not popped and recomputed if it is smaller than any previously computed marginal gain, thereby avoiding the need

to recompute for all data. The detailed algorithms of SimpleGreedy and LazyGreedy are included in App. B.1.

2.2 Data Subset Selection

Data subset selection (SS) is an important field in *machine learning* (ML) because data curators may not have the resources to acquire, store or train ML models with large amount of data in the entire data pool D. The goal of SS is thus to find a subset S of size at most k that results in an ML model as good as possible. There are three types of SS: (1) *supervised SS* where the labels of data are always known; (2) *unsupervised SS* where the labels of data are always unknown; (3) *active learning* where the label of a datum is known only after it is selected. In this report, we focus on (1) and discuss how a surrogate monotone submodular utility function that approximates the performance of ML model trained on each subset S can be designed so that efficient submodular maximization algorithms in Sec. 2.1 can be used[1]. Intuitively, the selected subset S should be *informative* (reduces the ML model's uncertainty) or *representative* (of the data pool D), which are important desiderata to be considered.

3 Offline Setting

In this section, we focus on SS in the traditional *offline* setting, where the entire data pool D is always accessible to the data curator. The problem definition is as follows: given a data pool D consisting of n data, select a subset $S \subset D$ of size at most k such that the resulting ML model achieves the highest possible performance. We define each datum $d_i \in D$ as $(\mathbf{x}_i, y_i)$, where $\mathbf{x}_i$ is its feature vector and y_i is its label.

3.1 Submodular Utility Functions

Different ML models favor different types of data. Thus, one needs to design specific submodular utility functions tailored to various ML models to approximate their behaviors well. In this section, we introduce submodular utility functions related to 3 classic ML models, namely *K-nearest neighbors* (KNN), *naïve Bayes* (NB) and *Gaussian process* (GP). While we will provide a basic explanation of each model in the main report, unfamiliar readers may refer to App. B.2 for more detailed descriptions.

3.1.1 K-Nearest Neighbors Utility Function

For any data $d_i, d_j \in D$, we may define a metric $m(i, j)$ that measures the distance between their features $\mathbf{x}_i$ and $\mathbf{x}_j$ (e.g., L^2-distance). A KNN model assumes that data close to each other should have similar labels, and thus predicts the label for a new observation x to be the most frequent label among its K nearest neighbors in the (selected) training set S, measured under m. When $K = 1$, this implies that the training set S is better (of higher utility) when each datum d (not necessarily in S) is closer to d's nearest neighbor in S with the same label. Based on this intuition, Wei et al. [22] proposes a submodular function f_{NN} based

[1]There are also other SS approaches such as *coreset* [1] and *score-based methods* like *influence function* [10], while we focus on *submodular maximization* methods in this report.

on the non-negative *similarity* metric $s(i,j) = \max_{d_{i'}, d_{j'} \in D} m(i',j') - m(i,j)$:

$$f_{\text{NN}}(S) \coloneqq \sum_{y \in \mathcal{Y}} \sum_{d_i \in D^y} \max_{d_j \in S \cap D^y} s(i,j),$$

where $\mathcal{Y}$ denotes the set of all possible labels and $D^y \subseteq D$ denotes the set of all data in the data pool D that are labelled as y. By choosing the subset S which maximizes the sum of pairwise similarity between S and D, optimizing f_{NN} selects the subset that is the most representative of the ground set. f_{NN} is clearly monotone because for any datum $d \in D$, adding an element to S will not decrease d's similarity to its closest neighbor in S with the same label. f_{NN} is also submodular because for any $S \subseteq T$, for any datum $d \in D$, d's similarity to its closest neighbor in T is no smaller than in S. When an element e is added to S and T, either e becomes the closest neighbor of d in $T \cup \{e\}$ (in which case the similarity increase caused by d for S is greater than or equal to T), or d's closest neighbor in T does not change (in which case the similarity increase for T caused by d is 0 and for S is non-negative). Since f_{NN} is monotone submodular, the subset S can be selected using the greedy algorithms as introduced in Sec. 2.1.

Remark 1. *If all labels y are the same, f_{NN} is equivalent as the famous facility location function [14].*

Remark 2. *Although f_{NN} is motivated by the KNN model, it also works for other ML models because it selects a representative subset. Similar reasonings apply to the remaining utility functions.*

3.1.2 Naïve Bayes Utility Function

Let the feature vector $\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \cdots & x_R \end{bmatrix}$. Let $p(x_r)$ denote the probability that the r-th feature X_r (as a discrete random variable) is x_r (for discrete features). Based on the Bayes' rule [21], $p(y|\mathbf{x}) \propto p(y)p(\mathbf{x}|y)$. The NB model further assumes that each feature X_r is conditionally independent to each other given the label $Y = y$, that is, $p(y)p(\mathbf{x}|y) = p(y)\prod_{r=1}^{R} p(x_r|y)$. Till this end, the NB model learns $p(y)$ as how often the label y appears in the training set S; it learns $P(x_r|y)$ as how often the feature value x_r appears in class y in the training set S. When a new observation $\mathbf{x}$ arrives, the NB model computes $p(y|\mathbf{x})$ for each label y and outputs the most probable label.

Assuming the distribution of each label in S is similar to that in D [22], a selected subset S is better (of higher utility) if the frequency of each x_r in each class y in S is similar to that in D. Thus, Wei et al. [22] proposes to measure the utility of S by considering a cross-entropy-like similarity between the counts of each (X_r, Y) in S and in D:

$$f_{\text{NB}}(S) \coloneqq \sum_{y \in \mathcal{Y}} \sum_{r=1}^{R} \sum_{x_r \in \mathcal{X}_r} |D^{x_r,y}| \log |S^{x_r,y}|,$$

where $\mathcal{X}_r$ is the set of all possible values for feature X_r, $D^{x_r,y}$ is the set of all data in D with feature $X_r = x_r$ that are labelled as y and $S^{x_r,y}$ is the set of such data in S. f_{NB} is clearly monotone because adding an element to S either increases $\log |S^{x_r,y}|$ or does not change it. f_{NB} is also submodular because it is a sum of concave of modular functions. Thus, one may use the greedy algorithms to efficiently select a good subset S using f_{NB} to train an NB model.

3.1.3 Gaussian Process Utility Function

Unlike classical ML models that fit a function to data, a GP model assumes that functions are sampled from a *Gaussian process* defined by a mean function and a covariance (kernel) function. Typically, it starts with a prior of 0 mean and large covariance. As more data is observed, the posterior reduces uncertainty and converges gradually to the true function. Therefore, the utility of training set S is higher when it reduces the GP model's uncertainty to a greater extent. Based on this, Mirzasoleiman et al. [15] proposes to use the mutual information function to measure the uncertainty (i.e., diversity) of S and how much observing S can reduce entropy of a GP model. It is defined as

$$f_{\mathrm{MI}}(S) = \log\det(\mathbf{I} + \gamma \mathbf{K}_S), \tag{1}$$

where γ is a scale parameter, $\mathbf{K}_S$ is a principal submatrix indexed by S of the positive semi-definite similarity kernal $\mathbf{K}$. f_{MI} is monotone because $\mathbf{K}$ is positive semi-definite and a large S would result in a larger determinant; f_{MI} is submodular because it is proportional to the mutual information between the data and the GP model.

3.2 Deletion-Robust Data Subset Selection via Submodularity

In Sec. 1, we have motivated why data from the data pool D might be deleted and the selected subset S should preserve high utility even after data deletions. We consider two settings below: (1) the data curator knows that the deletion size u is smaller than a constant U, but does not known which data will be deleted; (2) the data curator knows the deletion probability of each datum $d \in D$, but does not know how many or which data will be deleted.

3.2.1 Known Deletion Size

If the data curator uses any monotone submodular function f defined in Sec. 3.1 and knows that a specific set $\mathcal{U} \subseteq D$ with $|\mathcal{U}| \leq U$ is deleted, then the remaining utility of a set S, $f_{\mathcal{U}}(S) \coloneqq f(S \cap (D \setminus \mathcal{U}))$. $f_{\mathcal{U}}$ is still monotone submodular because of the *conditioning* property. To maximize the worst-case utility given any deletion set $\mathcal{U}$, we need maximize the following function:

$$f_{\mathrm{KDS}}(S) \coloneqq \min_{\substack{\mathcal{U} \subseteq D \\ |\mathcal{U}| \leq u}} f_{\mathcal{U}}(S),$$

with the cardinality constraints that $|S| \leq k$. Unfortunately, f_{KDS} is not monotone submodular. To address this, Krause et al. [11] considers another function

$$\bar{f}_c(S) \coloneqq \frac{1}{\Upsilon} \sum_{\substack{\mathcal{U} \subseteq D \\ |\mathcal{U}| \leq u}} \min\{f_{\mathcal{U}}(S), c\},$$

where c is a parameter (to be explained later) and Υ is the number of possible deletion sets $\mathcal{U}$ with $|\mathcal{U}| \leq U$. The following facts can be observed:

- $\bar{f}_c$ is submodular because it is the average of some truncated submodular functions. Hence, we can maximize $\bar{f}_c$ using the greedy algorithms;
- If $\max_{\substack{S\subseteq D\\|S|\leq k}} f_{\text{KDS}}(S) < c$, then $\max_{\substack{S\subseteq D\\|S|\leq k}} \bar{f}_c(S) < c$; otherwise, $\max_{\substack{S\subseteq D\\|S|\leq k}} \bar{f}_c(S) = c$.

Based on the second point above, we can maximize f_{KDS} by looking for the largest c such that $\max_{\substack{S\subseteq D\\|S|\leq k}} \bar{f}_c(S) = c$, and such a c is also the maximum of f_{KDS}. Note that for any value of c we can easily check whether $\max_{\substack{S\subseteq D\\|S|\leq k}} \bar{f}_c(S) = c$ using greedy algorithms because of the first bullet point. Therefore, to maximize f_{KDS}, we can initially set a very small $c_{\min}$ such that $\max_{\substack{S\subseteq D\\|S|\leq k}} \bar{f}_{c_{\min}}(S) = c_{\min}$ and a very large $c_{\max}$ such that $\max_{\substack{S\subseteq D\\|S|\leq k}} \bar{f}_{c_{\min}}(S) < c_{\max}$ and then perform a binary search over $c \in [c_{\min}, c_{\max}]$. The detailed algorithm is in Alg. 1. As we can see, maximizing the non-submodular function f_{KDS} is an interesting application of submodular optimization using greedy algorithms.

Algorithm 1 The SATURATION algorithm.

Input: $f_{\mathcal{U}}$ for $\mathcal{U} \subseteq D$ and $|\mathcal{U}| \leq u$
$c_{\min} \leftarrow 0; c_{\max} \leftarrow \min_i f_i(D); S^* \leftarrow \emptyset$
while $(c_{\max} - c_{\min}) \geq \frac{1}{m}$ **do** ▷ perform binary search
  $c \leftarrow (c_{\max} + c_{\min})/2$
  Define $\bar{f}_c(S) \triangleq \frac{1}{m}\sum_i \min\{f_i(S), c\}$ ▷ define a submodular helper function
  Find S by SIMPLEGREEDY with utility function $\bar{f}_c$
  if $|S| > \alpha k$ **then** ▷ when $\alpha = 1$, exactly k data are selected; larger α incurs better robustness
    $c_{\max} \leftarrow c$
  else
    $c_{\min} \leftarrow c; S^* \leftarrow S$
return S^*

Note that Alg. 1 selects αk ($\alpha \geq 1$) instead of k data. When $\alpha = 1$, exactly k data are selected but there is no theoretical guarantee; when $\alpha = 1 + \log \max_{d\in D} \sum_{\mathcal{U}} f_{\mathcal{U}}(\{d\})$ (i.e., the data curator is allowed to select more than k data), the post-deletion utility is at least as high as the post-deletion utility of any possible k-sized subset of D.

3.2.2 Known Deletion Probabilities

In this setting, we assume that the data curator knows the deletion probability of each datum in the selected subset S. For each S, let p_S denote the distribution of the remaining subset $A \subseteq S$ after data deletions. Sim et al. [20] proposes the following *(expected) deletion-anticipative data selection* objective:

$$f_{\text{DADS}}(S) \coloneqq \mathbb{E}_{A\sim p_S(\cdot)}[f(A)],$$

where f is any monotone submodular function defined in Sec. 3.1. Note that whether f_{DADS} is submodular depends on how the probability distribution p_S is modelled. We discuss two cases below:

- When the data curator models the deletion of each datum independently, he can set a deletion probability v_d for each datum $d \in D$. In this case, we can consider the probability distribution $P_D(\cdot)$ of remaining subsets over the entire data pool D such that for any subset $B \subseteq D$, $p_D(B) =$

$\prod_{d\in B}(1-\upsilon_d)\prod_{d\in D\setminus B}\upsilon_d$. We notice that $f_{\text{DADS}}(S) = \mathbb{E}_{B\sim p_D(\cdot)}[f(S\cap B)]$, and is thus submodular because it is a linear combination of conditioned submodular functions.

- When the data curator considers each datum to have an equal probability of deletion (but not necessarily independent) (i.e., they are *symmetric*), for each selected set size $s = 1, 2, \cdots, k$ he needs to set a probability $p_s(a)$ for each remaining set size $a = 0, 1, \cdots, s$ after deletions. Fortunately, in practice they only need to set $p_k(\cdot)$ (e.g., as a discrete uniform distribution) and then recursively compute other $p_s(\dot{c})$ (for $s = k-1, k-2, \cdots, 1$) following $p_s(a) = p_{s+1}(a) + p_{s+1}(a+1)$. In this setting, f_{DADS} is also submodular by changing the data pool D in the independent setting into sets of size k instead.

Since f_{DADS} is monotone submodular in such cases, we can use the greedy algorithms to perform data selection efficiently.

Remark 3. *The expectation in f_{DADS} can be estimated using sample average approximation [9].*

Remark 4. *Instead of expectation, Sim et al. [20] also considers using the conditional value-at-risk [18] (i.e., the expectation of worst-cases utilities under the unknown remaining set) instead of expectation in f_{DADS} so that the selected subset S is even more robust and f_{DADS} is still submodular.*

4 Streaming Setting

In real life, it is prohibitively expensive for data curators to store a large amount of data together. Moreover, some data comes naturally in the fashion of a stream (e.g., time-sensitive data such as stock prices). Therefore, in this section, we focus on SS in the *streaming* setting, where data from D arrives sequentially at the data curator. Let $d^{(t)}$ denote the datum arriving at timestamp t and the stream is thus $\left(d^{(1)}, d^{(2)}, \cdots, d^{(n)}\right)$. The goal remains to select the best set S of size at most k: $\max_{\substack{S\subseteq D\\ |S|\leq k}} f(S)$. The greedy algorithms need to access all elements for k rounds, so it requires k passes of the stream, which is unrealistic (e.g., time-sensitive data would only arrive once). Thus, we first introduce streaming submodular maximization algorithms in Sec. 4.1 that only requires a few or even one pass of the stream and can be used in together with the submodular utility functions defined in Sec. 3.1. Then, we discuss about how to make such algorithms deletion-robust in Sec. 4.2.

4.1 Streaming Submodular Maximization Algorithms

4.1.1 The StreamGreedy Algorithm

The StreamGreedy algorithm [7] is the first streaming algorithm for submodular maximization under cardinality constraint. The algorithm adopts a greedy yet different from SimpleGreedy approach. Referring to Alg. 2, it first selects the first k data from the stream, which makes S_k a valid solution (Line 2 to 5); after that, whenever a new datum d_t arrives at iteration t, it identifies the best k-sized subset of $S_{t-1}\cup\{d_t\}$ and set it to be S_t while dropping the remaining least valuable datum (Line 7 to 10). The process is continued until there is no significant improvement of S_t over S_{t-1} for NI consecutive rounds (Line 11 to 15). Clearly, the utility of S_t increases monotonically with number of iterations t until it halts.

Algorithm 2 The StreamGreedy algorithm.

Input: number of tolerated rounds without significant improvement NI; improvement threshold η

initialize $S_0 \leftarrow \emptyset$
for $t = 1, 2, \ldots, k$ **do**
  receive d_t
  $S_t \leftarrow S_{t-1} \cup \{d_t\}$ ▷ add the t-th incoming element to the currently selected set
$\rho \leftarrow 0$
while $t > k$ **and** $\rho < NI$ **do** ▷ check if there are already NI rounds without significant improvement
  receive d_t
  $e_t \leftarrow \arg\min_{e \in S_{t-1}} f(S_{t-1} \cup \{d_t\}) - f(S_{t-1} \cup \{d_t\} \setminus \{x\})$ ▷ find the current least valuable datum
  $S_t \leftarrow S_{t-1} \cup \{d_t\} \setminus \{e_t\}$ ▷ replace the last valuable datum by d
  if $f(S_t) - f(S_{t-1}) < \eta$ **then**
    $\rho \leftarrow \rho + 1$ ▷ increment ρ if there is no significant improvement
  else
    $\rho \leftarrow 0$ ▷ reset ρ if there is a significant improvement
return S_t

Unfortunately, the simple and intuitive algorithm does not have any theoretical guarantee unless some very strong assumptions are made. In fact, Badanidiyuru et al. [2] show that the StreamGreedy algorithm can have arbitrary bad performance.

4.1.2 The SieveStreaming Algorithm

Suppose we have already selected a subset A consisting of $a < k$ data and we aim for $f(S_{\text{OPT}})$ by the time we select k data, then the next element we select should bring a marginal gain of at least $((f(S_{\text{OPT}}) - f(A))/(k - a)$ (otherwise we "sieve" it) because the next marginal gain will be the largest among all future marginal gains due to submodularity. Based on this intuition, Badanidiyuru et al. [2] proposes the SieveStreaming streaming algorithm. There are 2 more considerations about the above intuition:

- Since there may only be one datum whose marginal gain is above the threshold $((f(S_{\text{OPT}}) - f(A))/(k - a)$ (`hard`) and all others are below it, plus it may be located at the end of the stream, it is not rational to discard all other data that are just below the threshold. Therefore, some lower threholds could be added, e.g., $((f(S_{\text{OPT}})/2 - f(A))/(k - a)$ (`soft`).
- There is no way to know S_{OPT}, otherwise we sould have already solved the problem. However, due to submodularity, we know that for any d_t, $f(\{d_t\}) \leq f(S_{\text{OPT}}) \leq k \cdot f(\{d_t\})$. That is, if $\gamma := \max_{d \in D} f(\{d\})$, then $\gamma \leq f(S_{\text{OPT}}) \leq k\gamma$.

Combining the two, all thresholds between $(\gamma/2 - f(A))/(k - a)$ (using `hard`) and $(k\gamma - f(A))/(k - a)$ (using `soft`) are suitable. Since it is intractable to try all thresholds within the range, the authors only consider the thresholds in the form $(1+\epsilon)^i$ for some $i \in \mathbb{Z}$. The rest of the algorithm is clear (see Alg. 3): it keeps track of the largest $f(\{d\})$ seen so far as γ; for each threshold $\omega \in \{(1+\epsilon)^i : i \in \mathbb{Z}; \gamma \leq (1+\epsilon)^i \leq 2k\gamma\}$, it maintains a selected subet S_ω and only adds a datum d_t if its marginal gain is not less than $(\omega/2 - f(A))/(k - a)$. After n iterations, the best subset S_{ω^*} amongst all S_ω is returned.

The SieveStreaming algorithm is a one-pass algorithm that is guaranteed to return a set S such that $|S| < k$ and $f(S) \geq (1/2 - \epsilon) f(S_{\text{OPT}})$. While the full proof can be found in App. C.1, a proof sketch is as

Algorithm 3 The SieveStreaming algorithm.

initialize thresholds $\Omega_0 \leftarrow \{(1+\epsilon)^i : i \in \mathbb{Z}\}$
$S_\omega \leftarrow \emptyset$ for $\omega \in \Omega_0$; $\gamma \leftarrow 0$
for $t = 1, 2, \cdots, n$ **do**
  $\gamma \leftarrow \max(\gamma, f\{d_t\})$
  $\Omega_t \leftarrow \{(1+\epsilon)^i : i \in \mathbb{Z}; \gamma \le (1+\epsilon)^i \le 2k\gamma\}$ ▷ parallel instances
  delete all S_ω whose $\omega \notin \Omega_t$
  for $\omega \in \Omega_t$ **do**
    if $\Delta_f(d_t|S_\omega) \ge \frac{\omega/2 - f(S_\omega)}{k - |S_\omega|}$ **and** $|S_\omega| < k$ **then** ▷ selection criteria
      $S_\omega \leftarrow S_\omega \cup \{d_t\}$
$\omega^* \leftarrow \arg\max_{\omega \in \Omega_n} f(S_\omega)$
return S_{ω^*}

follows:

- For any ω, if S_ω has exactly k elements, then the utility is at least $\omega/2$ because of how the marginal gain of each selected datum is thresholded.
- If S_ω has less than k elements, it means that the data in $S_{\text{OPT}} \setminus S_\omega$ are not selected and this means their marginal gain was once smaller than $\omega/2k$ (a quantity no smaller than the threshold). Since there are not more than k data in $S_{\text{OPT}} \setminus S_\omega$ and due to submodularity, $f(S_{\text{OPT}}) - f(S_{\omega^*}) \le k \cdot \omega/2k = \omega/2$.
- The largest ω is at least $(1-\epsilon) \cdot f(S_{\text{OPT}})$ because of how the thresholds are chosen. Therefore, we have $((1-\epsilon)/2) f(S_{\text{OPT}}) \le f(S_{\omega^*})$.

4.1.3 The SieveStreaming++ Algorithm

The SieveStreaming algorithm takes $O(k \log k/\epsilon)$ space because it needs to maintain $\log_{1+\epsilon} 2k\gamma - \log_{1+\epsilon} \gamma = O((\log k)/\epsilon)$ subsets consisting of $O(k)$ selected data. As a further improvement in the space complexity, Kazemi et al. [8] proposes the SieveStreaming++ algorithm. The intuition behind SieveStreaming++ is the same as that of SieveStreaming but with one additional desideratum: the recursively updated γ is an overly pessimistic lower bound of $f(S_{\text{OPT}})$. Instead, one can always maintain a separate lower bound LB based on the already selected S_ω's and use $\max\{\gamma, LB\}$ as the overall lower bound. Any new datum should bring a marginal gain of at least $\max\{\gamma, LB\}/2k$ in order to be added, otherwise its marginal gain is smaller than $f(S_{\text{OPT}})/2k$ and does not need to be added (`soft`). The detailed algorithm is in Alg. 4.

Because SieveStreaming++ and SieveStreaming have the same underlying mechanism, their approximation ratio is the same (i.e., $\frac{1}{2} - \epsilon$). However, SieveStreaming++ only needs $O(k/\epsilon)$ space, which can be shown by bounding LB more delicately. The detailed proof is in App. C.2.

4.2 Deletion-Robust Streaming Submodular Maximization

We have illustrated how the submodular utility functions in Sec. 3.1 for SS can be applied to the streaming setting via the streaming submodular maximization algorithms defined in Sec. 4.1. Could the deletion-robust data selection techniques in Sec. 3.2 be adapted to the streaming setting as well? The answer is yes. In particular, the setting with known deletion probabilities in Sec. 3.2.2 can be easily adapted because

Algorithm 4 The SieveStreaming++ algorithm.

$\gamma \leftarrow 0$; $\tau \leftarrow 0$; $LB \leftarrow 0$
for $t = 1, 2, \cdots, n$ **do**
  $\gamma \leftarrow \max(\gamma, f\{d_t\})$
  $\tau = \frac{\max(LB,\gamma)}{2k}$ ▷ a better lower bound
  discard all sets S_ω with $\omega < \tau$
  $\Omega_t \leftarrow \{(1+\epsilon)^i : i \in \mathbb{Z}; \tau/(1+\epsilon) \leq (1+\epsilon)^i \leq \gamma\}$ ▷ parallel instances
  for $\omega \in \Omega_t$ **do**
    if ω is a new threshold **then**
      $S_\omega \leftarrow \emptyset$
    if $\Delta_f(d_t|S_\omega) \geq \omega$ **and** $|S_\omega| < k$ **then** ▷ selection criteria
      $S_\omega \leftarrow S_\omega \cup \{d_t\}$; $LB \leftarrow \max\{LB, f(S_\omega)\}$
$\omega^* \leftarrow \arg\max_{\omega \in \Omega_n} f(S_\omega)$
return S_{ω^*}

the deletion-robust utility function f_{DADS} itself is monotone submodular and can be directly plugged into the algorithms in Sec. 4.1. Therefore, we instead consider the following two settings: (1) the data curator receives only data as a stream and does not known which data will be deleted after selection is done (i.e., offline deletions); (2) the data curator receives both data and deletion requests in the stream (i.e., streaming deletions).

4.2.1 Offline Deletions

In this setting, Mitrovic et al. [16] formulates the deletion-robust SS problem by defining a selected subset S as deletion-robust if for any deletion set $\mathcal{U} \subseteq D$ such that $|\mathcal{U}| \leq u$,

$$\max_{\substack{A \subseteq S \setminus \mathcal{U} \\ |A| \leq k}} f(A) \geq \alpha \cdot \max_{\substack{B \subseteq D \setminus \mathcal{U} \\ |B| \leq k}} f(B). \tag{2}$$

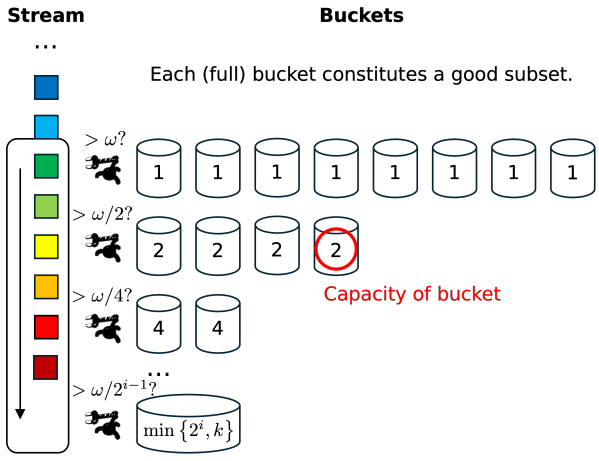

Figure 1: **Illustration of the Star-T algorithm.**

That is, for any deletion set $\mathcal{U}$, one can find a subset A of size at most k from the selected set S such that its utility is at least α of the best possible subset of size at most k in $D \setminus U$. Compared with the setting in Sec. 3.2.1 where we only want the worst set after deletions to be as good as possible (but the other set can become as bad as the worst set), now we want all sets after deletions to maintain robust utility.

To achieve this, an intuitive approach is to maintain multiple disjoint sets of data, each with reasonably high utility. When data deletions occur, only a proportion of these sets would be affected, while the rest would retain their utility. However, a key challenge is how to partition the data effectively to ensure each set maintains high utility. To address this, Mitrovic et al. [16] proposes STAR-T algorithm (Alg. 5) which works like a sorting system with filtered inserters in a factory, as illustrated in Fig. 1. Specifically, suppose ω is the final model utility we aim to obtain. This can be achieved by a single very good datum whose marginal gain to the empty set $\geq \omega$, or two good but not so good data whose marginal gains $\geq \omega/2$, or four even worse data whose marginal gains $\geq \omega/4$ and so on. Based on this, whenever a datum arrives in the datum, we assign it to the corresponding buckets based on the thresholds. When all data haave arrived, the STAR-T algorithm returns the union of all buckets as S^*.

During data deletions, data belonging to some buckets are removed, while data belong to other buckets are not. Therefore, to identify a subset from S^* with utility comparable to ω, the STAR-T algorithm simply needs to run the SIMPLEGREEDY or LAZYGREEDY algorithm to find a good subset.

When $\omega \geq \beta \cdot f(S_{\mathrm{OPT}})$, the STAR-T algorithm is guaranteed to satisfy Eq. 2 with $\alpha = 0.149(1 - 1/\log k)$, which approaches 0.149 as $k \to \infty$. The proof is similar to the intuition provided above, so we will direct it to App. C.3. To find such an ω, similar approaches to the SIEVESTREAMING algorithm can be used, that is, to run multiple instances of STAR-T in parallel and return the best output.

Algorithm 5 The STAR-T algorithm.

Input: estimated objective value ω; unit of buckets κ (i.e., buckets come in multiple of κ)
$B_{i,j} \leftarrow \emptyset$ for $0 \leq i \leq \lceil \log k \rceil, 1 \leq j \leq w\lceil k/2^i \rceil$ ▷ create all buckets
for $t = 1, 2, \ldots, n$ **do**
  for $i = 0, 1, \ldots, \lceil \log k \rceil$ **do**
    for $j = 1, 2, \ldots, \kappa\lceil k/2^i \rceil$ **do**
      if $|B_{i,j}| < \min\{2^i, k\}$ and $\Delta_f(d_t | B_{i,j}) \geq \omega / \min\{2^i, k\}$ **then** ▷ insert data to correct bucket
        $B_{i,j} \leftarrow B_{i,j} \cup \{d_t\}$
        break
$S = \bigcup_{i,j} B_{i,j}$
return S

Remark 5. *Recent follow-up works [6, 4] have explored deletion-robust submodular maximization in the same setting but under different constraints (matroids, knapsack).*

4.2.2 Streaming Deletions

In this setting, deletion requests arrive in the stream. At each timestamp t, the data curator either receive a datum d_t or receive a deletion request of a certain datum. For ease of notation, we use D_t to represent the set of data that has arrived at timestamp t and $\mathcal{U}_t$ to reprsent the set of data that has been deleted at timestamp t. Clearly, $D_1 \subseteq D_2 \subseteq \cdots$ and $\mathcal{U}_1 \subseteq \mathcal{U}_2 \subseteq \cdots$. As before, we assume that the maximum deletion

size is u, that is, $|\mathcal{U}_n| \leq u$. The goal is to maximize the utility at each timestamp t, that is,

$$\max_{\substack{S_t \in D_t \setminus \mathcal{U}_t \\ |S_t| \leq k}} f(S_t).$$

When there is no deletion, there are already streaming algorithms that achieve α-approximation such as the SieveStreaming algorithm in Sec. 4.1.2. We call any of such algorithm a GenericStreaming algorithm. Therefore, if we can convert the streaming problem with deletions to one without deletions, then the problem is solved. Based on this intuition, Mirzasoleiman et al. [15] proposes a cascading approach with $u+1$ GenericStreaming instances named the RobustStreaming algorithm (Alg. 6):

- When a new datum arrives, it will first be passed to the first GenericStreaming instance. The first instance may either select it or discard it (e.g., in SieveStreaming). To select the datum, the first instance may also discard some other data. We call these discarded data R.
- Instead of being directly discarded, data in R are instead passed to the second GenericStreaming instance (via the Add subroutine in Alg. 7); if the second instance consequently discards some data, then these data are passed to the third GenericStreaming instance; so on and so forth.
- When a deletion request arrives, we first find which instance the deleted datum belongs to. After that, we nullify that instance and cascade all the leftover data from that instance to the next instance, so on and so forth (via the Delete subroutine in Alg. 8).
- In the end, we return the output of the first non-null instance.

Since there are $u+1$ instances in total, there is always at least one non-null instance in the end. If the GenericStreaming algorithm used gives an α-approximation, it is provable that the output returned by the RobustStreaming algorithm is also an α-approximation. This is because the first non-null GenericStreaming instance has seen all the data in D_t (either the earlier instance discarded them or they met a deleted datum and cascaded all the remaining data) and thus it will give an α-approximation.

Algorithm 6 The RobustStreaming algorithm. Note that it uses the Add subroutine in Alg. 7 and the Delete subroutine in Alg. 8.

Input: data stream D_t; deletion set $\mathcal{U}_t$ with $|\mathcal{U}_t| \leq u$; number of GenericStreaming instances $\nu \leq u+1$

$t \leftarrow 0$
$M_t^{(i)} \leftarrow 0, S_t^{(i)} \leftarrow \emptyset$ for $i = 1, 2, \ldots, \nu$

while $\{D_t \setminus D_{t-1}\} \cup \{\mathcal{U}_t \setminus \mathcal{U}_{t-1}\} \neq \emptyset$ **do**
  if $\{\mathcal{U}_t \setminus \mathcal{U}_{t-1} \neq \emptyset\}$ **then** ▷ a deletion request is received
    $d_t \leftarrow \{\mathcal{U}_t \setminus \mathcal{U}_{t-1}\}$
    Delete(d_t)
  else ▷ a datum is received
    $d_t \leftarrow \{D_t \setminus D_{t-1}\}$
    Add$(1, d_t)$
  $t = t + 1$
  $S_t = \{S_t^{(i)} | i = \min\{j \in \{1, 2, \ldots, \nu\}, M_t^j \neq \text{null}\}\}$
return S_t

Algorithm 7 The ADD subroutine.

Input: GENERICSTREAMING instance index i; incoming dataset R

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

Algorithm 8 The DELETE subroutine.

Input: deletion request to delete d

for $i \in \{1, 2, \ldots, \nu\}$ **do**
  if $d \in M_t^{(i)}$ **then**
    $R_t^{(i)} = M_t^{(i)} \setminus \{d\}$
    $M_t^{(i)} \leftarrow$ null
    $\text{Add}(i+1, R_t^{(i)})$
    return

5 Conclusion

In this report, we present how submodular optimization serves as powerful tool for SS. We introduce submodular utility functions tailored to specific ML models, which enable data curators to efficiently select informative or representative subsets via greedy algorithms. We also extend this to the streaming setting and present streaming submodular maximization algorithms, which can be used together with the submodular utility functions for SS. We also address a practical challenge where data can be deleted from data pool or selected subset, and show how submodularity can still be leveraged to select deletion-robust subsets.

References

[1] Pankaj K Agarwal, Sariel Har-Peled, Kasturi R Varadarajan, et al. Geometric approximation via coresets. *Combinatorial and computational geometry*, 52(1):1–30, 2005.

[2] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Streaming submodular maximization: Massive data summarization on the fly. In *Proc. KDD*, pages 671–680, 2014.

[3] Cedric Burton, Laura De Boel, Christopher Kuner, Anna Pateraki, Sarah Cadiot, and Sára G Hoffman. The final European Union General Data Protection Regulation. *BNA Privacy & Security Law Report*, 15:153, 2016.

[4] Shuang Cui, Kai Han, and He Huang. Deletion-robust submodular maximization with knapsack constraints. In *Proc. AAAI*, volume 38, pages 11695–11703, 2024.

[5] Pedro Domingos and Geoff Hulten. Mining high-speed data streams. In *Proc. KDD*, pages 71–80, 2000.

[6] Paul Duetting, Federico Fusco, Silvio Lattanzi, Ashkan Norouzi-Fard, and Morteza Zadimoghaddam. Deletion robust submodular maximization over matroids. In *Proc. ICML*, pages 5671–5693. PMLR, 2022.

[7] Ryan Gomes and Andreas Krause. Budgeted nonparametric learning from data streams. In *Proc. ICML*, volume 1, page 3. Citeseer, 2010.

[8] Ehsan Kazemi, Marko Mitrovic, Morteza Zadimoghaddam, Silvio Lattanzi, and Amin Karbasi. Submodular streaming in all its glory: Tight approximation, minimum memory and low adaptive complexity. In *Proc. ICML*, pages 3311–3320. PMLR, 2019.

[9] Anton J Kleywegt, Alexander Shapiro, and Tito Homem-de Mello. The sample average approximation method for stochastic discrete optimization. *SIAM Journal on optimization*, 12(2):479–502, 2002.

[10] Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *Proc. ICML*, pages 1885–1894. PMLR, 2017.

[11] Andreas Krause, H Brendan McMahan, Carlos Guestrin, and Anupam Gupta. Robust submodular observation selection. *Journal of Machine Learning Research*, 9(12), 2008.

[12] Hui Lin and Jeff A Bilmes. How to select a good training-data subset for transcription: submodular active selection for sequences. In *Interspeech*, pages 2859–2862, 2009.

[13] Michel Minoux. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques: Proceedings of the 8th IFIP Conference on Optimization Techniques Würzburg, September 5–9, 1977*, pages 234–243. Springer, 2005.

[14] Pitu B Mirchandani and Richard L Francis. *Discrete location theory*. 1990.

[15] Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. Deletion-robust submodular maximization: Data summarization with "the right to be forgotten". In *Proc. ICML*, pages 2449–2458. PMLR, 2017.

[16] Slobodan Mitrovic, Ilija Bogunovic, Ashkan Norouzi-Fard, Jakub M Tarnawski, and Volkan Cevher. Streaming robust submodular maximization: A partitioned thresholding approach. *Proc. NeurIPS*, 30, 2017.

[17] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. An analysis of approximations for maximizing submodular set functionsâI. *Mathematical programming*, 14:265–294, 1978.

[18] Sergey Sarykalin, Gaia Serraino, and Stan Uryasev. Value-at-risk vs. conditional value-at-risk in risk management and optimization. In *State-of-the-art decision-making tools in the information-intensive age*, pages 270–294. Informs, 2008.

[19] Alexander Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

[20] Rachael Hwee Ling Sim, Jue Fan, Xiao Tian, Patrick Jaillet, and Bryan Kian Hsiang Low. Deletion-anticipative data selection with a limited budget. In *Proc. ICML*, 2024.

[21] James V Stone. Bayes' rule: a tutorial introduction to Bayesian analysis. 2013.

[22] Kai Wei, Rishabh Iyer, and Jeff Bilmes. Submodularity in data subset selection and active learning. In *Proc. ICML*, pages 1954–1963. PMLR, 2015.

[23] Andreas Weiler, Marc H Scholl, Franz Wanner, and Christian Rohrdantz. Event identification for local areas using social media streaming data. In *Proceedings of the ACM SIGMOD Workshop on Databases and Social Networks*, pages 1–6, 2013.

[24] Yunyue Zhu and Dennis Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 358–369. Elsevier, 2002.

A Project Administration

A.1 Brief Summary of Contributions

Overall, the two authors have contributed equally to the report.

- The **Abstract**, **Introduction** and **Conclusion** sections were written together by the two authors.
- For the **Offline Setting** section, Fan Jue focused on the submodular utility functions and the Known Deletion Probabilities case of deletion-robust data subset selection; Tian Xiao focused on the Known Deletion Size case of deletion-robust data subset selection. However, both authors proofread and made appropriate updates on the scripts written by each other.
- For the **Streaming Setting** section, Fan Jue focused on the SieveStreaming algorithm and Tian Xiao focused on the StreamGreedy, SieveStreaming++, Star-T and RobustStreaming algorithms. However, both authors proofread and made appropriate and significant updates on the scripts written by each other.

A.2 Declaration

This report was entirely written by us. We have not used this material as part of any other previous assessment (at NUS or elsewhere), and we will not use them for any assessments in other courses we are taking this semester.

B Additional Backgrounds

B.1 Submodularity

Alg. 9 gives the pseudocode of the SimpleGreedy algorithm.

Algorithm 9 The SimpleGreedy algorithm

initialize $S \leftarrow \emptyset$
while $|S| \leq k$ **do** $d^* = \arg\max_{d \in D \setminus S}[\Delta_f(d|S)]$
    $S \leftarrow S \cup \{d^*\}$
return S

As a further improvement in runtime, the LazyGreedy algorithm has been proposed, where the marginal gains of elements are stored in a priority queue. At each iteration t, the element with the top marginal gain in the priority queue is popped. If it has not been recomputed based on the currently selected subset, then it is recomputed and put back into the queue; otherwise, it is selected into the subset S. The LazyGreedy algorithm avoids the need to compute the marginal gain of every element at every iteration and thus becomes more efficient in practice.

Algorithm 10 The LazyGreedy algorithm.

initialize a priority queue $PQ = \{\}$ and for each data $d \in D$, insert $(d, f(\{d\}))$ into PQ ordered by decreasing order of $f(\{d\})$
$(d^*, \Delta_f(d^*|S = \emptyset)) \leftarrow$ the $d^* \in D$ which gives the highest $f(d^*)$
while $|S| \leq k$ **do**
    $(d, \Delta_f(d|S)) \leftarrow \text{pop}(PQ)$
    if $\Delta_f(d|S) \geq \Delta_f(d^*|S)$ **then**
        $S \leftarrow S \cup \{d\}$
    else
        $(d^*, \Delta_f(d^*|S)) \leftarrow (d, \Delta_f(d|S))$
return S

B.2 Machine Learning Models

B.2.1 K-Nearest Neighbors

Given the train set S and a query data $\mathbf{x}$, a KNN model finds the K nearest neighbor of $\mathbf{x}$ in S using similarity measure $s(\cdot, \cdot)$ and outputs the most frequent label as the prediction. Let $V_S(\mathbf{x})$ be the K data closest (measured by $s(\cdot, \cdot)$) to $\mathbf{x}$ in set S. More formally, the prediction mechanism is

$$\hat{y} = \arg\max_y |\{d : d \in V_S(\mathbf{x}) \wedge y_d = y\}|.$$

B.2.2 Naïve Bayes

An NB model is a probablistic classifier based on Baye's theorem with the "naïve" assumption that all features are conditionally independent given the class label. NB starts with a prior belief on the distribution of each class label, $p(y)$. For each data observed, each feature updates the posterior belief by $p(y|x_i) = \frac{p(x_i|y)p(y)}{p(x_i)}$. When NB receives a query for the label of $\mathbf{x} = (x_1, \ldots, x_R)$, the prediction is the class with the highest posterior probability:

$$\hat{y} = \arg\max_y p(y) \prod_{i=1}^{R} p(x_i|y),$$

where the numerator $\prod_{i=1}^{R} p(x_i)$ is a constant and thus omitted.

B.2.3 Gaussian Process

A GP model is a regression model which assumes that functions are sampled from a *Gaussian process* defined by a mean function and a covariance (kernel) function. Specifically, the kernel function $\mathbf{K}(\mathbf{x}, \mathbf{x}')$ measures the similarity between $\mathbf{x}$ and $\mathbf{x}'$. Given any set S, the matrix $\mathbf{K}_S(\mathbf{x}, \mathbf{x}')$ indexed by data in S is symmetric and positive semidefinite.

Just like other probabilistic models, a GP model learn by updating its posterior using Bayes' rule [21]. During learning, its uncertainty is gradually decreased.

C Proofs

C.1 Theoretical Guarantees for SieveStreaming Algorithm

At the end of the algorithm, there are two cases:

1. S_{ω^*} has k elements. We have $f(S_{\omega^*}) \geq \frac{\omega}{2k} \cdot |S_{\omega^*}| = \frac{\omega}{2} \geq (1-\epsilon) \cdot f(S_{\text{OPT}})$.
2. $|S_{\omega^*}| < k$. For each $d_j \in S_{\text{OPT}} \setminus S_{\omega^*}$, suppose it is rejected when the current solution was $S_j \subseteq S_{\omega^*}$. Then we have
$$\Delta_f(d_j|S_j) \leq \frac{\omega^*/2 - f(S_j)}{k - |S_j|}, \text{ and } f(S_j) \geq \frac{\omega^*|S_j|}{2k}.$$
Thus we have $\Delta_f(d_j|S_j) \leq \frac{\omega^*}{2k}$ which gives $\Delta_f(d_j|S_{\omega^*}) \leq \frac{\omega^*}{2k}$ due to submodularity. Therefore,
$$\begin{aligned} f(S_{\text{OPT}} \cup S_{\omega^*}) - f(S_{\omega^*}) &\leq \sum_{d_j \in S_{\text{OPT}} \setminus S_{\omega^*}} f(S_j \cup \{d_j\}) - f(S_j) \\ &\leq \sum_{d_j \in \in S_{\text{OPT}} \setminus S_{\omega^*}} \frac{v}{2k} \leq \frac{v}{2} \leq \frac{f(S_{\text{OPT}})}{2} \\ \implies f(S_{\text{OPT}}) - f(S_{\omega^*}) &\leq \frac{\text{OPT}}{2} \\ \implies f(S_{\omega^*}) &\geq \frac{1}{2} f(S_{\text{OPT}}). \end{aligned}$$

□

C.2 Theoretical Guarantees for SieveStreaming++ Algroithm

Since all sets S_ω with $\omega < \tau$ are discarded, the optimal $f(S_{\text{OPT}})$ is at least $LB = \max_\omega(f(S_\tau)) \geq \max_\omega(|S_\omega| \times \omega)$. For any remaining $\omega \geq \frac{LB}{2k}$, we have $|S_\omega| \leq \frac{LB}{\omega}$. The case $\frac{LB}{2k} \leq \omega \leq \frac{LB}{k}$ is trivial. For $\omega \geq \frac{LB}{k}$, the upper bound on the size of S_ω decreases geometrically with the first term equal to k as ω increases (with a fixed LB). Thus the memory complexity is

$$O\left(\left\lceil \frac{k \log 2}{\epsilon} \right\rceil + \sum_{i=0}^{\log_{1+\epsilon} k} \frac{k}{(1+\epsilon)^i}\right) = O\left(\frac{k}{\epsilon}\right).$$

□

C.3 Theoretical Guarantees for Star-T Algorithm

The proof for the theoretical guarantees of Star-T algorithm is divided into two cases: (i) there exists a partition in S such that all of its buckets are full; (ii) there does not exist such as partition where at least half of its buckets are full.

Case (i): Let i^* be one of such partitions. Every partition contains $w\lceil k/2^i \rceil$ buckets, and there are at least $wk/2^{i^*+1}$ full buckets in partition i^*. Thus we have for any full bucket $B_{i^*,j}$,

$$|B_{i^*,j} \cap \mathcal{U}| \leq \frac{2^{i^*+1}u}{wk} \implies |B_{i^*,j} \setminus \mathcal{U}| \geq |B_{i^*,j}| - \frac{2^{i^*+1}u}{wk}.$$

Since every new datum increases the value of the bucket $B_{i^*,j}$ by at least $\omega/2^{i^*}$ and $B_{i^*,j}$ is full, we have datum in this bucket brings at least $\omega/|B_{i^*,j}|$ gain. Thus, we have

$$f(B_{i^*,j} \setminus \mathcal{U}) \geq \left(|B_{i^*,j}| - \frac{2^{i^*+1}u}{wk}\right) \frac{\omega}{|B_{i^*,j}|} = \omega\left(1 - \frac{2^{i^*+1}u}{|B_{i^*,j}|wk}\right),$$

which reduces to

$$f(B_{i^*,j} \setminus \mathcal{U}) \geq \omega(1 - \frac{4u}{wk}),$$

since $2^{i^*+1} \leq 4|B_{i^*,j}$.Thus after applying SimpleGreedy algorithm, we have the theoretical guarantee

$$f(S) \geq (1 - e^{-1})(1 - \frac{4u}{wk})\omega$$

Case (ii): Let $i > 0$ be any partition, B_i be a bucket in i. Given the previous partition's bucket B_{i-1}, it has $|B_{i-1}| < 2^{i-1}$ because it is not full. The loss in B_i from B_{i-1} is no more than

$$f(B_i \cap \mathcal{U}|B_{i-1}) \leq \sum_{d \in B_i \cap \mathcal{U}} f(d|B_{i-1}) < \sum_{d \in B_i \cap \mathcal{U}} \frac{\omega}{2^{i-1}} = \frac{\omega}{2^{i-1}}|B_i \cap \mathcal{U}|, \tag{3}$$

since when $f(d) < \frac{\omega}{2^{i-1}}$, $f(d|B_{i-1}) < f(e)$; when $f(d) > \frac{\omega}{2^{i-1}}$, we have the same inequality as $|B_{i-1}| < 2^{i-1}$ or the algorithm would add d to B_{i-1}.

Lemma 1. *For any $i \geq 1$, we have*

$$f\left(B_i \cap \mathcal{U}| \bigcup_{j=1}^{i-1}(B_j \setminus \mathcal{U})\right) \leq \sum_{j=1}^{i} \frac{\omega}{2^{j-1}}|B_j \cap \mathcal{U}_j| \tag{4}$$

When $i = 1$ and B_0 is not full (i.e. $|B_0| = 0$), Eq. (3) gives $f(B_1 \cap \mathcal{U} = f(B_1 \cap \mathcal{U}|B_0) \leq |B_1 \cap \mathcal{U}|\frac{\omega}{2^0}$. Then Lem. 1 can be proven inductively.

Then by submodularity and Lem. 1, we have

$$\begin{aligned} f\left(\bigcup_{i=1}^{\lceil \log k \rceil}(B_i \setminus \mathcal{U})\right) &\geq f(B_{\lceil \log k \rceil}) - f\left(B_{\lceil \log k \rceil} \cap \mathcal{U}| \bigcup_{i=0}^{\lceil \log k \rceil - 1}(B_i \setminus \mathcal{U})\right) \\ &\geq f(B_{\lceil \log k \rceil}) - \sum_{i=1}^{\lceil \log k \rceil} \frac{\omega}{2^{i-1}}|B_i \cap \mathcal{U}|, \end{aligned}$$

where each $|B_i \cap \mathcal{U}|$ can be upper bounded by $\frac{2^{i+1}|B_i \cap \mathcal{U}|}{wk}$. Hence $\sum_{i=1}^{\lceil \log k \rceil} \frac{\omega}{2^{i-1}}|B_i \cap \mathcal{U}| \leq \sum_{i=1}^{\lceil \log k \rceil} \frac{\omega}{2^{i-1}} \frac{2^{i+1}|B_i \cap \mathcal{U}|}{wk} \leq \frac{4|\mathcal{U}|}{wk}\omega$. Putting together, we can simplify the equation above as

$$f\left(\bigcup_{i=1}^{\lceil \log k \rceil}(B_i \setminus \mathcal{U})\right) \geq f(B_{\lceil \log k \rceil}) - \frac{4|\mathcal{U}|}{wk}\omega.$$

Besides, since $\bigcup_{i=0}^{\lceil \log k \rceil} |B_i \setminus \mathcal{U}| \leq \bigcup_{i=0}^{\lceil \log k \rceil} |B_i| \leq k + \bigcup_{i=0}^{\lceil \log k \rceil} 2^i \leq 3k$, we have

$$f(\mathrm{OPT}(3k, S \setminus \mathcal{U})) \geq f\left(\bigcup_{i=1}^{\lceil \log k \rceil} (B_i \setminus \mathcal{U})\right) \geq f(B_{\lceil \log k \rceil}) - \frac{4|\mathcal{U}|}{wk}\omega.$$

Hence, we can complete the proof with

$$f(S^*) = f(\textsc{Greedy}(k, S \setminus \mathcal{U})) \geq (1 - e^{-1/3}) f(\mathrm{OPT}(3k, S \setminus \mathcal{U})) \geq (1 - e^{-1/3}) \left(f(B_{\lceil \log k \rceil}) - \frac{4u}{wk}\omega\right).$$

□