STOCHASTIC SUBMODULAR DATA FORGETTING

by

Ramón Rico Cuevas

Supervisor: Prof. Dr. Yannis Velegrakis
Co-supervisor: Prof. Dr. Arno Siebes

A thesis submitted in conformity with the requirements
for the degree of Master of Science in Computing Science

Department of Information and Computing Sciences
Utrecht University

Universiteit Utrecht Sharing science, *shaping tomorrow*

STOCHASTIC SUBMODULAR DATA FORGETTING

Ramón Rico Cuevas
Master of Science in Computing Science

Department of Information and Computing Sciences
Utrecht University
2023

Abstract

Our ability to collect data is rapidly surpassing our ability to store it. As a result, organizations are faced with difficult decisions about what data to retain, and in what form, to meet their business goals while complying with storage restrictions. We address this retention issue in the context of relational data by exploiting continuous stochastic submodular maximization technology. Given a relational dataset D, a query-log Q, and a budget B, the objective of our problem, data forgetting, is finding a subset $D^* \subseteq D$ with at most B tuples, such that it is still possible to compute, based solely on D^*, approximate answers to the expected workload of queries. We formalize data forgetting as a stochastic subset selection exercise and consider two variants: independent (ignoring the topology of D), and dependent (taking it into account). Independent (resp. Dependent) data forgetting can be cast an instance of the 0-1 Knapsack problem (resp. the discrete stochastic submodular maximization problem). Capitalizing on this connections, two routines are proposed: `IndepDF` and `DepDF`. The former solves the independent variant exactly in pseudo-polynomial time, and the latter approximately solves, in expectation, the dependent variant within a $(1 - 1/e - \epsilon)$ factor. Experimentally, `IndepDF` (resp. `DepDF`) it's shown to fetch high quality solutions when Q retrieves homogeneous (resp. heterogeneous) views of D. Furthermore, time performance-wise, both routines are mildly affected by increases in $|D|$, not affected at all by increases in $|Q|$, and completely unaffected by increases in B. This almost-perfect independence wrt the input's size, allows them to operate in data intensive environments.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the past two decades, **Big Data** has become the prism through which we look at almost every aspect of the world. Data-driven solutions have enabled revolutionary advances in critical industries, profoundly transforming our lives. Big Data is characterized for being varied, very large and generated at an ever-growing speed. Nonetheless, it's raw and therefore must be stored and subsequently analyzed to extract knowledge out of it.

To date, advances in storage technology have allowed organizations to accumulate data with almost no restriction. However, it's estimated that the size of the global datasphere (i.e., the digital universe) will surpass storage production by an order of magnitude as soon as 2025 [Reinsel et al.(2018)]. Moreover, as pointed out by the recent General Data Protection Regulation (GDPR) [EU(2016)], uncontrolled data storage could jeopardize the privacy and security of individuals. To mitigate this danger, such law confers any resident of a protected region the right to enforce a company the deletion of his personal data.

Consequently, the luxury of unconstrained data storage is coming to an end, forcing data-driven enterprises and research institutes to face the **data reduction problem**: retaining the information hidden in the data while respecting regulatory, storage, and processing constraints [Milo(2019)]. In other words, data reduction involves deciding what data to keep and in what form, to comply with legal data regulations and storage restrictions, while minimizing information loss. Furthermore, due to the vast and ever-growing nature of Big Data, automating the reduction exercise becomes vital to avoid flooding.

This research tackles the data reduction problem in the context of (traditional) relational data by exploiting (less traditional) **stochastic submodular maximization** technology. We formalize data reduction as an optimization subset selection exercise which we've named **data forgetting**. Given a relational database D and a query-log Q, we aim at finding a succinct subset of tuples $D^* \subseteq D$ (storage constraint) that allow to faithfully answer queries sampled from the underlying distribution $\mathcal{Q}$ which generated query-log Q (processing constraint). That is, subset D^* retains the information hidden in D when the former is understood as the ability to cope with the expected query workload.

Example 1.0.1. *Consider the toy database shown in table 1.1. Assume that of the three tuples in D we can only keep two (storage constraint). Furthermore, suppose that the only information we have about the usage of D is query-log*

$$Q = \{q_1 : \textit{Age} = 25, \quad q_2 : \textit{City} = \textit{Amsterdam} \ \wedge \ \textit{Job} = \textit{Teacher}, \quad q_3 : \textit{Name} = \textit{Olivia}\}.$$

That is, three unique queries occurring with equal probability, each retrieving a different part of D (processing constraint). Query q_1 retrieves the complete database, q_2 retrieves d_1 and d_3, and q_3 retrieves d_3. In other words, the answer set of q_1 (resp. q_2, q_3) in D, denoted $q_1(D)$ (resp. $q_2(D), q_3(D)$), is equal to $\{d_1, d_2, d_3\}$ (resp. $\{d_1, d_3\}, \{d_3\}$). Consequently, d_3 is accessed three times, d_1 is accessed twice, and d_2 is accessed once.

id	Name	Last name	Age	City	Job	Married	Children
d_1	James	Smith	25	Amsterdam	Teacher	Yes	2
d_2	Wade	Brown	25	Paris	Firefighter	No	0
d_3	Olivia	Smith	25	Amsterdam	Teacher	Yes	2

Table 1.1: Toy database $D = \{d_1, d_2, d_3\}$.

Considering this information, which two tuples should be kept? Given that $|D| = 3$, there exist only $\binom{3}{2} = 3$ possible combinations. Namely,

$$D' = \{d_1, d_3\}, \quad D'' = \{d_1, d_2\}, \quad D''' = \{d_2, d_3\}.$$

Arguably, the most natural election would be D' (i.e, keeping the two most frequently accessed tuples). However, by making this choice, we are overlooking the structure of the data. That is, the fact that tuples in the database may be similar or dissimilar to each other.

It's natural to conceive D as a graph G_D where each tuple corresponds to a node, and there exist a weighted edge between any two nodes reflecting their degree of similarity (see figure 1.1). Furthermore, the level of fulfillment of any given query q should take into account the fact that if a requested tuple d is not retained, its information may be totally/partially encompassed by another retained tuple d' that's similar to the former.

Interestingly, upon taking into account the topology of the data and measuring query satisfiability in the aforementioned way, subset D''' (i.e, a subset containing the least frequently accessed datapoint!) turns out to be the best possible election when maximizing the expected query workload fulfillment. Intuitively, this is because tuples d_1 and d_3 are very similar to each other and very dissimilar to d_2. Hence, every answer set $q(D), q \in Q$ is more ***diversely covered*** *by a subset containing d_2 and one of d_1 or d_3 exclusively.*

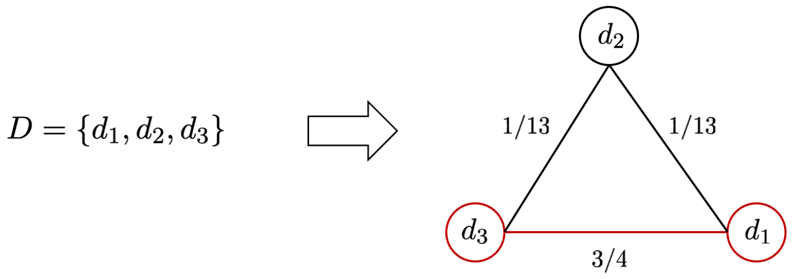


Figure 1.1: Graph representation of database $D = \{d_1, d_2, d_3\}$ as in table 1.1, denoted G_D. Each node corresponds to a tuple in D. An edge exists between any two nodes if the (Jaccard) similarity between the corresponding tuples (when understood as sets) is strictly positive. For example, tuples d_1 and d_3 are identified with sets

$$\mathfrak{Constrs}(d_1) = \{\text{Name = James},\ \text{Last name = Smith},\ \text{Age = 25},\ \text{City = Amsterdam, Job = Teacher, Married = Yes, Children = 2}\},$$

$$\mathfrak{Constrs}(d_3) = \{\text{Name = Olivia},\ \text{Last name = Smith},\ \text{Age = 25},\ \text{City = Amsterdam, Job = Teacher, Married = Yes, Children = 2}\}$$

respectively. Consequently,

$$Jaccard(d_1, d_3) = \frac{|\mathfrak{Constrs}(d_1) \cap \mathfrak{Constrs}(d_3)|}{|\mathfrak{Constrs}(d_1) \cup \mathfrak{Constrs}(d_3)|} = 6/8 = 3/4,$$

depicted in red.

We considered two variants of the data forgetting problem: **independent** and **dependent**. In the former, independence between the tuples' content is assumed, and consequently, query fulfillment is understood as **query precision**. For the contrary, the latter takes into account the structure of the data as described by means of the toy example, and hence, query fulfillment is measured via the **facility location** function.

Independent data forgetting can be proven to be an instance of the **0-1 Knapsack** problem. As a consequence, computational complexity-wise, independent data forgetting lies in **weak-NP** and can be exactly solved in pseudo-polynomial time. On the other hand, dependent data forgetting can be cast as an instance of the **stochastic submodular maximization** problem under a **coverage function** (namely, the facility location function). Maximization of submodular coverage functions can be reduced to **max-cover**, which belongs to strong-NP and it's NP-hard to approximate beyond $(1 - 1/e)$. Therefore, regarding the computational complexity, dependent data forgetting lies in strong-NP and it's NP-hard to approximate beyond $(1 - 1/e)$.

We developed two data forgetting routines with strong theoretical guarantees to address both variants of the problem: `IndepDF` and `DepDF`. `IndepDF` exactly solves independent data forgetting by resorting on the 0-1 Knapsack solver. Furthermore, `DepDF` approximately solves dependent data forgetting with a factor of $(1 - 1/e - \epsilon)$, in expectation, by: lifting the problem into the continuous domain using **multilinear continuation**; solving the continuous version of problem with **Stochastic Continuous Greedy**, a continuous stochastic submodular maximization algorithm; and last, maping the continuous solution back into the discrete domain via **randomized pipage rounding**. The extensive experimental evaluation on both real and synthetic data confirm the theoretical guarantees, and showcase the feasibility of the proposed routines in data intensive environments.

Our contributions can be summarized as follows.

1. We formulate the data forgetting problem (a unifying framework) and it's two most natural variants.
2. We study the theoretical computational complexity of both variants and prove their NP-hardness.
3. We propose two very efficient and highly scalable algorithms (`IndepDF` and `DepDF`) to solve the two variants of the problem.
4. We present an extensive experimental evaluation, based on real and synthetic data, demonstrating the effectiveness, efficiency, and scalability of our proposed data forgetting routines.

Outline. Chapter 2 provides an extensive literature review on data reduction, covering data summarization and the seminal work on of what we have named data forgetting. Chapter 3 formally introduces the data forgetting problem in its two flavours as well as the `IndepDF` and `DepDF` routines. Finally, chapter 4 presents the experimental results and some concluding remarks.

Chapter 2

Related Work

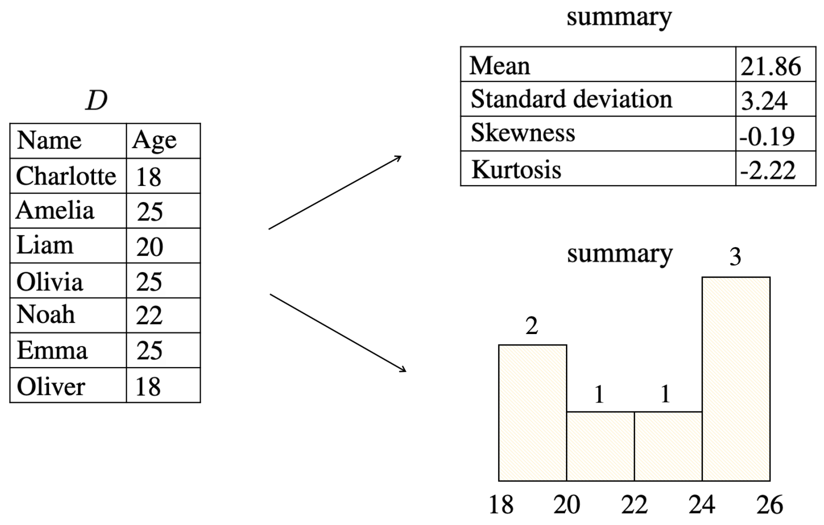
Data preservation is the act of conserving and maintaining the safety and integrity of data over time [Divjak(2022)]. An important affair that arises in the context of Big Data preservation is **data reduction**. Data reduction involves retaining the information hidden in massive data while respecting storage, regulatory, and processing constraints [Milo(2019)]. In other words, if a dataset has become too large to be stored, data reduction seeks deciding what data to keep and in what form to minimize the loss of information while meeting storage restrictions and complying with legal data regulations.

It is worth noting that legal data regulations should always be followed independent of the ability or inability to store a given dataset. Regulatory constraints simply force the retention or deletion of certain data records over time (e.g., the "right to be forgotten", appearing in article 17 of the General Data Protection Regulation (GDPR) [EU(2016)], confers any resident of a protected region the right to require a company the deletion of his personal data within one month of the formal request). Consequently, regulatory constraint enforcement is a non-algorithmic task within the data reduction exercise.

Algorithmic data reduction essentially boils down to minimizing information loss while adhering to storage constraints. The simplest way of meeting a strict storage budget in a data intensive environment is to discard most of the data at the source where it is produced. Despite being a bad practice, upstream data discarding is still a commonly used procedure. Sensors in scientific instruments frequently transmit with smaller rates than what they are capable of, and devices like security cameras or sensor networks oftentimes resort on ad hoc decision rules to throw away most of the received input [Kersten and Sidirourgos(2017), Milo(2019)]. However, blindly rejecting data at the source often leads to information loss, as valuable observations may be injudiciously discarded. **Data reduction routines** deal with this shortcoming by seeking simultaneous minimization of storage needs and information loss. This chapter aims to survey the main techniques the scientific community has proposed to address the data reduction problem to date.

2.1 Data summarization

A **summary** of a dataset D is a brief synopsis that has a modest size compared to that of the latter. Summaries come in one of two flavours: as subsets of the original dataset D, or as collections of objects that jointly retain the fundamental traits of D. That is, as reduced collections of the original dataset (see figure 2.1b), or as representative values that replace the original data records (see figure 2.1a).



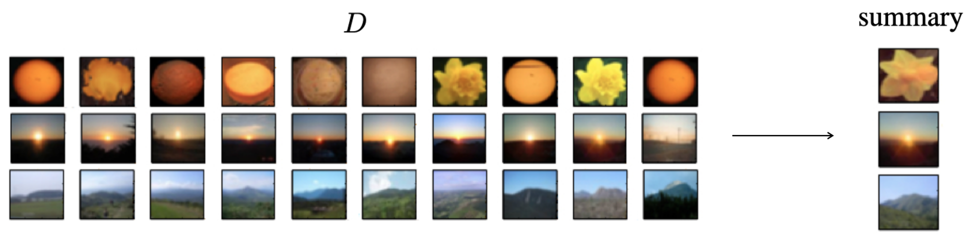
D

Name	Age
Charlotte	18
Amelia	25
Liam	20
Olivia	25
Noah	22
Emma	25
Oliver	18

summary

Mean	21.86
Standard deviation	3.24
Skewness	-0.19
Kurtosis	-2.22

(a) Summary consisting of representative values that replace the records in the original dataset.



(b) Summary consisting of a reduced collection of the original dataset. That is, a subset of the original dataset.

Figure 2.1: Illustrative example of both summary types. (a) Example of relational data summarization. Given a dataset D containing the name and age of 7 students (left), two summaries can be extracted. The first one is an assembly of summary statistics (upper right), and the second one is a histogram (lower right). We see that both the collection of aggregates and the histogram replace the original records by representative values that convey global information about the data. (b) Example of image collection summarization extracted from [Mirzasoleiman(2017)]. Given a dataset of 30 unique images D (left), the summary is a subset of 3 images that is as representative as possible of the complete dataset (right). The particular summary shown in the figure is the result of applying the GREEDI algorithm [Mirzasoleiman et al.(2013)] (see sec 2.1.2.5) to dataset D.

Following, we provide an overview of existing summarization techniques, each of which, given a dataset D, yield an outcome that falls within one of the two aforementioned categories.

2.1.1 Statistical data summarization

The first group of techniques we'll dive into are those that rely on statistics to create a summary of a given dataset. Three approaches stand out: **aggregation**, **histograming**, and **sampling**.

2.1.1.1 Aggregation

Arguably, the simplest form a summary can take is being a collection of constants $\alpha \in \mathbb{R}$. Given a dataset D, aggregation involves replacing the original dataset with an assembly of **summary statistics**. That is, numerical constants that convey information about the central tendency, dispersion or shape of the dataset's distribution [Upton and Cook(2002)].

The aggregation process is very simple: Fixed a dataset D (e.g., containing names and ages of students as in the example figure depicted in figure 2.1a), the records are replaced by summary statistics like the **arithmetic mean**, **standard deviation**, **skewness** or **kurtosis**. Following, we recall each one of these concepts.

Arithmetic mean The arithmetic mean is a measure of central tendency. Given a finite dataset $D = \{d_1, \dots, d_n\}$ consisting of n numeric records, the arithmetic mean, denoted μ, is the sum of the entries divided by the total number of them. That is

$$\mu = \frac{1}{n}\sum_{i=1}^{n} d_i.$$

In our running example, the numerical values d_i correspond to the age of the students. Hence, as shown in figure 2.1a, the mean $\mu = 21.86$.

Standard deviation The standard deviation is a measure of dispersion that indicates how far away from the mean the values in the dataset tend to lie. A small standard deviation indicates that the values concentrate around the mean, while a large standard deviation indicates that the values are spread out far away from the mean. Given a finite dataset $D = \{d_1, \dots, d_n\}$ made up of n numeric values, the standard deviation, denoted σ, is the square root of the sum of squared differences between each value and the mean. That is,

$$\sigma = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(d_i - \mu)^2}.$$

In our running example, as shown in figure 2.1a, the standard deviation $\sigma = 3.24$.

Skewness The skewness indicates the degree of asymmetry that a numeric dataset has with respect to its mean. It takes a positive value if the datapoints are concentrated towards the left of the mean, a negative value if the datapoints are concentrated towards the right of the mean, and it's equal to 0 if there exist approximately the same amount of values at both sides of the mean. Mathematically, given a numeric dataset $D = \{d_1, \ldots, d_n\}$, it's skewness, denoted $\tilde{\mu}_3$, is defined as the sum of differences between each value and the mean to the power of 3 divided by the total number of values times the standard deviation to the power of 3. Formally,

$$\tilde{\mu}_3 = \frac{\frac{1}{n}\sum_{i=1}^{n}(d_i - \mu)^3}{\sigma^3}.$$

In our running example, since there is one more datapoint to the right of the mean than to the left, the skewness takes a negative value. Namely, $\tilde{\mu}_3 = -0.19$.

Kurtosis The kurtosis reflects the likelihood that a given numeric dataset contains outlier values. Those datasets whose underlying distribution has slow asymptotically-approaching zero tails (positive kurtosis), will produce more outliers than those whose tails quickly vanish (negative kurtosis). Fixed a numeric dataset $D = \{d_1, \ldots, d_n\}$, the kurtosis, denoted $\tilde{\mu}_4$, has a similar expression to that of the skweness. Namely,

$$\tilde{\mu}_4 = \frac{\frac{1}{n}\sum_{i=1}^{n}(d_i - \mu)^4}{\sigma^4}.$$

In our running example, as can be observed from the **histogram** in the lower right of figure 2.1a, the tails from the underlying distribution seem to vanish very close to the mean. Consequently, the skewness has a negative value. Namely, $\tilde{\mu}_4 = -2.22$.

2.1.1.2 Histogramming

A more intricate version of the idea behind aggregation is histogramming. **Histograms** are simple tools that allow to compactly represent large volumes of data. A histogram summarizes a dataset by splitting it along any attribute (or attribute group) into a set of buckets. For each, a small set of summary statistics that approximately represent the data in such bucket is computed. The most basic version of this technique, divides a single attribute's domain into equi-width buckets and simply keeps the total number of data points that fall within each one [Ahmed(2019)].

A good example is depicted in the lower right of figure 2.1a. The input dataset D containing the name and age of students, is divided along the "age" attribute, assumed to range between 18 and 26, into the following equi-width buckets: $[18, 20), [20, 22), [22, 24), [24, 26)$. Subsequently, the frequency of datapoints belonging to each one is computed and presented in form of vertical bars of said height. The first interval contains 2 students, the second and third contain 1, and the fourth contains 3. As can be appreciated from this example, a histogram is a type of summary that provides insights into the shape of the dataset's underlying distribution. Furthermore, in combination with carefully chosen aggregates, it provides an almost unequivocally characterization of such distribution using low storage space.

Despite its cost-effectiveness, histogramming and aggregation alone could lead to information loss. In fact, after summarizing a dataset using these techniques, any query that aims at retrieving specific datapoints will return an empty answer when ran against the summary. Nevertheless, when combined with **sampling**, powerful procedures have been developed for efficiently computing approximate answers to complex queries over the retained synopsis [Orr et al.(2020)].

2.1.1.3 Sampling

Sampling involves selecting a representative subset (or **sample**) S from a given dataset $D = \{d_1, \dots, d_n\}$ via a stochastic mechanism. Sampling techniques include: **simple random, systematic, stratified**, and **clustering/multistage sampling** [Cochran(1977)]. Next, a brief description of each one is provided for completeness.

Simple random sampling Given a dataset D and a sample size $s \in \mathbb{N}$, sample S is built by selecting s datapoints $d_i \in D$ with equal probability (i.e., with uniform probability $\frac{1}{n}$).

As an example, consider the image dataset D displayed in figure 2.1b. Summarizing D via a simple random sample with sample size $s = 3$ amounts to drawing 3 images out of the 30 that conform D, each with a probability of $1/30$.

Systematic sampling Given a dataset D and a sample size $s \in \mathbb{N}$, an element $d_j \in D$ is chosen at random. Starting from this element, the dataset is split into s equal-length intervals. That is, D is divided into subsets $\{d_j, \dots, d_{j+L}\}, \{d_{j+L}, \dots, d_{j+2\cdot L}\}, \dots, \{d_{(n-1)/L}, \dots, d_{n/L}\} \subseteq D$, such that

$$\{d_j, \dots, d_{j+L}\} \cup \{d_{j+L}, \dots, d_{j+2\cdot L}\} \cup \dots \cup \{d_{(n-1)/L}, \dots, d_{n/L}\} = D,$$

where $L = n/s$, and all indices are taken modulo n. Thereafter, S is built by randomly choosing one element from inside each one of the s subsets. In other words, any element within each subset has equal probability $\frac{1}{L}$ of being chosen for inclusion in the sample S.

Going back to our running example, if $s = 3$ and d_j happens to be the left uppermost picture in dataset D (assumed to be ordered top-down and from left to right), the 3 equal-length intervals will be each 10-image row in D (see figure 2.2 for a visual representation). The systematic sample is then built by selecting one image from inside each row with uniform probability $1/10$.

Stratified sampling Given a dataset $D = \{d_1, \dots, d_n\}$ and a sample size $s \in \mathbb{N}$, D is partitioned into s disjoint subsets $G_1, \dots, G_s \subseteq D$ called **stratas**. Subsequently, the sample S is obtained by randomly selecting one element from within each strata G_i with probability $\frac{1}{|G_i|}$.

The essential difference between systematic and stratified sampling stems from the way dataset D is partitioned. In the former, the partition is always created in the same "systematic" manner starting from a seed element $d_j \in D$. For the contrary, there is no requirement with respect to the partition creation procedure in the latter. Refer to figure 2.2 for an example.

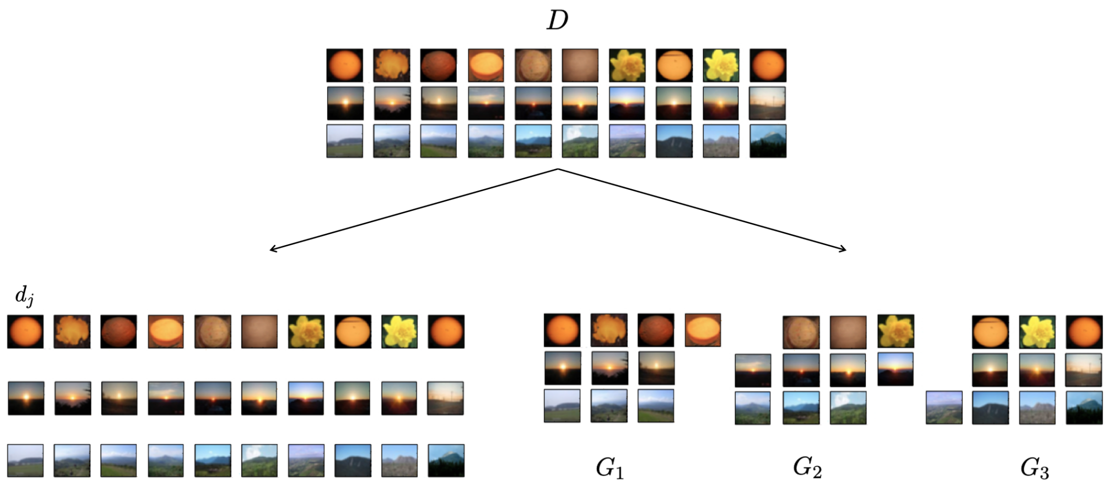


Figure 2.2: Systematic vs stratified sampling. On the left, the figure shows the 3 equal-length intervals corresponding to the systematic sampling scheme in the case where d_j is the left uppermost picture in the image dataset D. The sample of size 3 is obtained by choosing one element from within each interval. On the right, the figure shows an example of 3 possible stratas, denoted G_1, G_2, G_3, corresponding to the stratified sampling scheme. The sample of size 3 is built by selecting one element from inside each strata.

Clustering/Multi-stage sampling Given a dataset $D = \{d_1, \ldots, d_n\}$ and a sample size $s \in \mathbb{N}$, the dataset is organized into a fixed number of groups (clusters) $G_1, \ldots, G_k \subseteq D$ such that $\bigcup_{i=1}^{k} D_i = D$. Subsequently, $m < k$ clusters are selected from $G_1, \ldots, G_k$ with probability $\frac{1}{k}$. Following, the sample S is obtained by randomly selecting s/m elements inside of each of the m groups.

Similar to stratified sampling, multi-stage sampling partitions the dataset into subsets without any constraint on the subset creation process. However, instead of generating a sample from such groups directly, it first selects a smaller pool of subsets from which to build the sample S.

Due to its simplicity, sampling is a popular choice for data reduction. Furthermore, as aforestated, it enables approximate query answering when combined with techniques like aggregation and histogramming. Nevertheless, as a consequence of its stochastic nature, it does not allow controlling the characteristics of the selected subset. In fact, two runs over the same dataset of any sampling technique could (in general) yield outcomes with antipodal features. A deterministic alternative that crystalizes the sought subset traits in the form of an objective is optimization-based subset selection, often cast as a submodular maximization task.

2.1.2 Submodular data summarization

In order to summarize a dataset D via a subset $D^* \subseteq D$ in a deterministic fashion, it's vital defining a notion of subset utility. That is, a function $f : \mathcal{P}(D) \mapsto \mathbb{R}$ which measures the amount of representativeness that lies within each subset $D' \in \mathcal{P}(D)$. This way, given a budget $B \in \mathbb{N}$, the summarization task boils down to choosing a subset $D^* \subseteq D$ with cardinality at most B that reaches the highest utility. That is,

$$D^* = \underset{D' \subseteq D,\, |D'| \leq B}{\arg\max} f(D').$$

Sadly, this optimization exercise is intractable for arbitrary functions. However, as we will shortly see, in the case of **non-negative monotone submodular** objectives, there exists a very simple approximation algorithm that enjoys strong theoretical guarantees.

Definition 2.1.1. *Fixed D, a set function* $f : \mathcal{P}(D) \mapsto \mathbb{R}$ *is:*

· ***non-negative*** *if* $\forall\, A \subseteq D,\ f(A) \geq 0.$

· ***monotone*** *if* $\forall\, A \subseteq B \subseteq D\,, f(A) \leq f(B).$

· ***submodular*** *if* $\forall\, A, B \subseteq D,$

$$f(A \cap B) + f(A \cup B) \leq f(A) + f(B).$$

Equivalently, f *is* ***submodular*** *if* $\forall\, A \subseteq B \subseteq D$, *and* $d \in D \setminus B$,

$$\Delta_f(d|B) \leq \Delta_f(d|A),$$

where $\Delta_f(d|D') := f(D' \cup \{d\}) - f(D')$ *is the* ***discrete derivative*** *of* f *at* $D' \subseteq D$ *with respect to d.*

∎

The second definition of submodularity reveals a very intuitive interpretation of this property: given nested subsets $A \subseteq B \subseteq D$ and a datapoint $d \in D \setminus B$, the marginal gain of including d in the bigger one (i.e., in B) can't be greater than the marginal gain of including it in the smaller one (i.e., in A). That is, the larger the subset, the smaller the marginal gain provided by adding a new datapoint d to it. Therefore, submodularity is at heart a diminishing returns property that resembles a discrete version of concavity [Krause and Golovin(2014)].

2.1.2.1 Summarization objectives

Set functions that capture desirable features in a summary such as diversity and coverage over D are generally non-negative, monotone, and submodular. Intuitively, given $D'' \subseteq D' \subseteq D$, since D' contains D'', the former should have at least the same overall diversity and coverage as the latter (monotonicity). For the same reason, fixed $d \in D \setminus D'$, it is more likely to encounter a datapoint that's similar to d within D' than within D''. Consequently, adding d to D'' should provide greater marginal gain than including it in D' (submodularity).

Following, we present some examples of non-negative monotone submodular mappings that have been used as objectives in scene summarization [Simon et al.(2007)], document summarization [Lin and Bilmes(2012)], and image collection summarization [Tschiatschek et al.(2014)].

· **Facility location**: Given $D' \subseteq D$,

$$f_{\text{facility location}}(D') := \sum_{d \in D} \max_{d' \in D'} \text{sim}(d, d'),$$

where $\text{sim}(\cdot, \cdot)$ measures the similarity between any two elements of D. That is, the utility of $D' \subseteq D$ is the sum of similarities between each element in D and its closest neighbour inside D'.

· **Sum coverage**: Given $D' \subseteq D$,

$$f_{\text{sum coverage}}(D') := \sum_{d \in D} \sum_{d' \in D'} \text{sim}(d, d'),$$

where $\text{sim}(\cdot, \cdot)$ measures the similarity between any two elements of D. That is, the utility of $D' \subseteq D$ is the sum of all pairwise similarities between elements in D and D'.

· **Thresholded sum coverage**: Given $D' \subseteq D$,

$$f_{\text{thresh. sum coverage}}(D') := \sum_{d \in D} \min \left\{ \sum_{d' \in D'} \text{sim}(d, d'), \alpha \cdot \sum_{d' \in D'} \text{sim}(d, d') \right\},$$

where $\alpha \in \mathbb{R}$, and $\text{sim}(\cdot, \cdot)$ measures the similarity between any two elements of D. The utility of a subset $D' \subseteq D$ is measured in a similar fashion as by the sum coverage function. However, in this case, the inner sum is thresholded with the purpose of preventing any element $d \in D$ from being overly covered by D'.

· **Cluster diversity**: Given $D' \subseteq D$,

$$f_{\text{cluster diversity}}(D') := \sum_{j=1}^{k} g(D' \cap P_j),$$

where $g(\cdot)$ is a non-negative monotone submodular function, and $P_1, \ldots, P_k$ are different clusters yield by a clustering algorithm. According to this measure, subsets $D' \subseteq D$ that have the most diversity over the different clusters are those with the highest utility.

2.1.2.2 The greedy algorithm

As previously mentioned, cardinality-constrained set function maximization is intractable for arbitrary mappings. Even though the task remains NP-hard in the case of submodular non-negative monotone objectives, the GREEDY algorithm (see algorithm 1) provides a solution with $\left(1 - \frac{1}{e}\right)$ approximation guarantee to the optimal one in polynomial time [Nemhauser et al.(1978)].

Algorithm 1: GREEDY

```
Data: Dataset $D$, submodular monotone objective $f : \mathcal{P}(D) \mapsto \mathbb{R}_+$, and budget $B \in \mathbb{N}$.
Result: Subset $D^* \subseteq D$ such that $|D^*| = B$ and $f(D^*) \geq (1 - \frac{1}{e}) \cdot OPT$, where $OPT$ is
        the value of the optimal (intractable) solution.
$D_0 \leftarrow \emptyset$;
for $t = 1, \dots, B$ do
    $d_t \leftarrow \arg\max_{d \in D} \left( f(D_{t-1} \cup \{d\}) - f(D_{t-1}) \right)$;
    $D_t \leftarrow D_{t-1} \cup \{d_t\}$;
    $D \leftarrow D \setminus \{d_t\}$;
end
return $D^* = D_B$
```

The GREEDY algorithm strikes for its simplicity. The solution is built up incrementally from the empty set through a series of rounds. At round t, an element $d_t \in D$ that maximizes the marginal gain $\Delta_f(d|D_{t-1})$ gets included in the solution (breaking ties arbitrarily). Despite its simplicity, GREEDY requires $\mathcal{O}(n \cdot B)$ function evaluations to extract a summary D^* of size B out of a dataset D of size n. Consequently, if n is large or evaluating f is costly, running this procedure will be computationally expensive. To address this shortcoming, faster variants of the greedy algorithm have been developed.

2.1.2.3 The lazy greedy algorithm

The LAZY-GREEDY algorithm [Minoux(1978)] (see algorithm 2), exploits the submodularity of the objective to speed up GREEDY. The algorithm keeps an order-descending list of size $n = |D|$ where each element is an upper bound δ_d (initially ∞) on the marginal gain of a datapoint $d \in D$. At iteration t, LAZY-GREEDY computes $\Delta_f(d^*|D_{t-1})$ for the datapoint d^* corresponding to the head of the list, and updates δ_{d^*} to hold such value. If after the update δ_{d^*} is still the head of the list (i.e., $\delta_{d^*} \geq \delta_d \; \forall \, d \neq d^*$), then the submodularity of f guarantees that d^* is the element with the largest marginal gain. For the contrary, if after the update $\exists \, d \in D$ such that $\delta_{d^*} < \delta_d$, δ_{d^*} is inserted back into the list and the process is repeated.

Algorithm 2: `LAZY-GREEDY`

```
Data: Dataset $D$, submodular monotone objective $f : \mathcal{P}(D) \mapsto \mathbb{R}_+$, and budget $B \in \mathbb{N}$.
Result: Subset $D^* \subseteq D$ such that $|D^*| = B$ and $f(D^*) \geq (1 - \frac{1}{e}) \cdot OPT$, where $OPT$ is
        the value of the optimal (intractable) solution.
$D_0 \leftarrow \emptyset$; forall $d \in D, \delta_d \leftarrow \infty$;
for $t = 1, \ldots, B$ do
    forall $d \in D \setminus D_{t-1}$, $cur_d \leftarrow$ False;
    while True do
        $d^* \leftarrow \arg\max_{d \in D \setminus D_{t-1}} \delta_d$;     /* Get $d^*$ st $\delta_{d^*}$ head of the list */
        if $cur_{d^*}$ then
            $D_t \leftarrow D_{t-1} \cup \{d^*\}$;
            break
        end
        else
            $\delta_{d^*} \leftarrow f(D_{t-1} \cup \{d_t\}) - f(D_{t-1})$;     /* Update $\delta_{d^*}$ */
            if $(\delta_{d^*} \geq \delta_d \ \ \forall d \in D \setminus D_{t-1})$ then
                $cur_{d^*} \leftarrow$ True;     /* $\delta_{d^*}$ is still head of the list */
            end
        end
    end
end
return $D^* = D_B$
```

It is important to remark that although lazy evaluation enables `LAZY-GREEDY` to run orders of magnitude faster than `GREEDY` in practical scenarios, both manifest the same (superlinear) computational complexity. In fact, it is unknown the number of function evaluations performed by `LAZY-GREEDY` in arbitrary cases, but it is certain that it matches that of `GREEDY` in the worst one.

2.1.2.4 The stochastic greedy algorithm

The `STOCHASTIC-GREEDY` algorithm [Mirzasoleiman et al.(2015)] (see algorithm 3), was the first linear-time approximation algorithm for cardinality-constrained monotone submodular set function maximization. `STOCHASTIC-GREEDY` is a randomized routine capable of producing a solution with $\left(1 - \frac{1}{e} - \epsilon\right)$ approximation guarantee, in expectation, to the optimal one in $\mathcal{O}\left(n \cdot \log\left(\frac{1}{\epsilon}\right)\right)$ time. That is, in linear time in the size of the database $|D| = n$ and independent of the cardinality constraint B.

The crux behind the speed up lies in the inclusion of randomness through the use of sampling. Similar to `GREEDY`, the algorithm starts with an empty set and builds the solution incrementally by adding a new datapoint at each iteration. In contrast with `GREEDY`, the algorithm embeds a degree of randomness by including a sub-sampling step at the beginning of each round. At the start of round t, a subset R of size $s := \frac{n}{B} \cdot \log\left(\frac{1}{\epsilon}\right)$ is sampled from $D \setminus D_{t-1}$ uniformly at random (i.e., via simple random sampling, see section 2.1.1.3). Subsequently, an element $d_t \in R$ that maximizes the marginal gain $\Delta_f(d|D_{t-1})$ gets included in the solution (breaking ties arbitrarily).

Algorithm 3: STOCHASTIC-GREEDY

```
Data: Dataset $D$, submodular monotone objective $f : \mathcal{P}(D) \mapsto \mathbb{R}_+$, and budget $B \in \mathbb{N}$.
Result: Subset $D^* \subseteq D$ such that $|D^*| = B$ and $\mathbb{E}[f(D^*)] \geq (1 - \frac{1}{e} - \epsilon) \cdot OPT$, where
        $OPT$ is the value of the optimal (intractable) solution.
$D_0 \leftarrow \emptyset$;
for $t = 1, \ldots, B$ do
    $R \leftarrow$ a subset obtained by sampling $s$ random elements from $D \setminus D_{t-1}$;
    $d_t \leftarrow \arg\max_{d \in R} (f(D_{t-1} \cup \{d\}) - f(D_{t-1}))$;
    $D_t \leftarrow D_{t-1} \cup \{d_t\}$;
end
return $D^* = D_B$
```

To further boost the time performance of STOCHASTIC-GREEDY in practical scenarios, lazy evaluation can be leveraged. The lazy version of the algorithm maintains an order descending list of size $n = |D|$ where each element is an upper bound δ_d (initially ∞) on the marginal gain of a datapoint $d \in D$. At iteration t, the algorithm samples a subset R of size $s := \frac{n}{B} \cdot \log\left(\frac{1}{\epsilon}\right)$, and computes $\Delta_f(d^*|D_{t-1})$ for the datapoint $d^* \in R$ closest to the top of the list (instead of $\Delta_f(d|D_{t-1}), \forall d \in R$). Subsequently, δ_{d^*} is updated to hold such value. If after the update, δ_{d^*} remained in the same position within the list (i.e., $\delta_{d^*} \geq \delta_d \; \forall d \neq d^*$ s.t. $d \in R$), then the submodularity of objective f guarantees that d^* is the element with the largest marginal gain inside the set R. Again, as in the case of GREEDY, lazy evaluation does not improve the time complexity but often leads to faster executions in practice.

2.1.2.5 The price of scaling up, distributed and streaming paradigms

Despite producing good approximate solutions, the GREEDY algorithm and its accelerated variants (LAZY-GREEDY and STOCHASTIC-GREEDY) lack scalability. That is, their execution often becomes infeasible when running in data intensive environments. The impracticability of polynomial time algorithms in large scale data settings has motivated the scientific community to explore alternatives like the **distributed** and **streaming** paradigms.

The distributed paradigm A feasible way of maximizing a cardinality-constrained submodular objective in a data intensive context is to distribute the data among several machines and exploit parallel computation models like **MapReduce**.

MapReduce [Dean and Ghemawat(2004)], is a style of computing that has been implemented in several systems, including the popular open-source Apache **Hadoop** [Apache Hadoop(2005)]. Any implementation of MapReduce can be used to manage many parallel computations in a hardware-fault tolerant way. All the user has to do is write two functions: **Map** and **Reduce**. Data **chunks** from the surrounding **distributed file system** (DFS) are processed in parallel within independent machines via **map tasks** that produce key-value pairs. The way such key-value pairs are generated is determined by the user code written inside the Map function. Thereafter, pairs are grouped by key and feed to the **reduce tasks** which combine all values associated with each one. The way such combination is carried out is utterly determined by the user code written inside the Reduce function. After termination, the outcome of the MapReduce job is saved back into the surrounding DFS.

A simple distributed protocol for maximizing monotone submodular functions under cardinality constraints is GREEDI [Mirzasoleiman et al.(2013)]. The GREEDI algorithm (algorithm 4), schematically depicted in figure 2.3, is a two-round parallel routine that's easily implementable using MapReduce style computations. First, dataset D is partitioned uniformly at random into disjoint subsets $D_1, \dots, D_m$ that get distributed over m independent machines. Second, each machine $i \in \{1, \dots, m\}$ performs a first round of greedy selection over D_i (in parallel) to obtain solution set D_i^*. Third, solution sets are merged into $D' = \cup_{i=1}^m D_i^*$, and a second round of greedy selection is performed over D' yielding D_0^*. Last, the highest-scoring set D^* among the $m+1$ produced solutions gets returned.

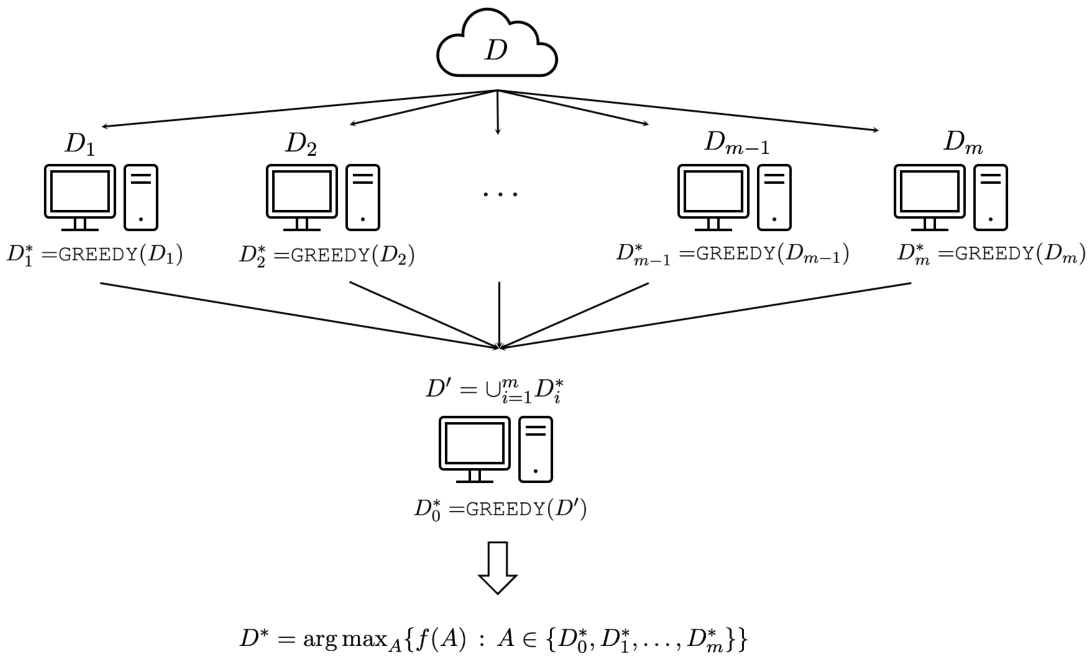


Figure 2.3: Illustration of GREEDI: D is the dataset, f is a non-negative monotone submodular function to be optimized, B is the maximum budget, m the number of independent workers, and D^* is the solution produced by the algorithm.

The GREEDI algorithm yields a solution with $\frac{1-1/e}{2}$ approximation guarantee in expectation to the optimal one [Barbosa et al.(2015)]. However, it's important to highlight that in case dataset D exhibits certain geometric structure and objective f belongs to the class of **Lipschitz-continuous** functions, a sharper bound can be derived. Namely, the approximation guarantee will approach $1 - \frac{1}{e}$ in expectation as $|D| = n \mapsto \infty$. Consequently, under such hypothesis, GREEDI attains a competitive solution to the intractable optimum in data intensive scenarios.

```
Algorithm 4: GREEDI
  Data: Dataset $D$, submodular monotone objective $f : \mathcal{P}(D) \mapsto \mathbb{R}_+$, budget $B \in \mathbb{N}$,
        number of machines $m \in \mathbb{N}$.
  Result: Subset $D^* \subseteq D$ such that $|D^*| = B$ and $\mathbb{E}[f(D^*)] \geq \frac{1-1/e}{2} \cdot OPT$, where $OPT$
          is the value of the optimal (intractable) solution.

  Partition $D$ into $m$ subsets $D_1, \ldots, D_m$ uniformly at random;
  Run GREEDY on each $D_i$ (in parallel) producing solutions $D_i^*$ of size $B$ for $i = 1, \ldots, m$;
   /* Observation 1:  GREEDY can be replaced by any of its
   accelerated versions like LAZY-GREEDY or STOCHASTIC-GREEDY.
   */
  Merge the resulting sets $D' \leftarrow \cup_{i=1}^{m} D_i^*$;
  Run GREEDY on $D'$ to obtain solution $D_0^*$ of size $B$;        /* Observation 1.  */
  return $D^* = \arg\max_A \{f(A) : A \in \{D_0^*, D_1^*, \ldots, D_m^*\}\}$;
```

The streaming model The use of **streaming algorithms** is another strategy to successfully maximize cardinality-constrained submodular functions in big data environments. When the dataset D becomes too large to store in memory, algorithms that pass once through the data constructing the solution set on the fly, have proven to be an effective alternative to distributed routines for massive data summarization.

In the **streaming model**, depicted in figure 2.4, the input dataset is assumed to arrive as a **data stream**. That is, as a partitioned permutation of D that's presented to the algorithm at times $t \in \{1, \ldots, T\}$ in form of a set succession $D_1, D_2, \ldots, D_T$. Depending on the nature of such permutation, the data stream is said to be **random** or **adversarial**. In the random setting, D is permuted uniformly at random, and in the adversarial one, D is permuted without restrictions by an evil "adversary/enemy". At each time step t, the streaming routine must decide without knowledge about the future which datapoints $d \in D_t$ should be added to the partial summary residing in memory. This way, the algorithm sequentially summarizes the input dataset on the fly by traversing it exactly once.

Streaming cardinality-constrained monotone submodular function maximization was addressed for the first time by SIEVESTREAMING [Badanidiyuru et al.(2014)]: A simple $1/2 - \epsilon$ approximation streaming protocol capable of extracting a summary D^* of size B out of a dataset D by exclusively using $\mathcal{O}(B \cdot \log(B/\epsilon))$ memory (i.e., independent of the dataset's size). Moreover, such algorithm attains the best possible approximation factor in the adversarial stream context. However, a sharper approximation factor can be reached in the random setting [Norouzi-Fard et al.(2018)]. Namely, by the MONOTONESTREAM algorithm [Liu et al.(2021)]. Such procedure is the fist streaming routine for cardinality-constrained monotone submodular function maximization able to guarantee a $(1 - \frac{1}{e} - \epsilon)$ approximation using $\mathcal{O}(B/\epsilon)$ memory. Again, independent of the dataset's size but larger than that required by SIEVESTREAMING.

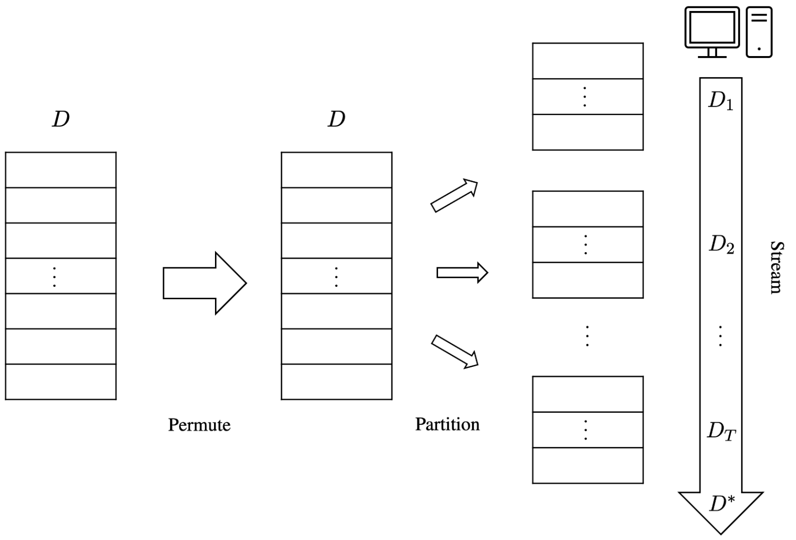


Figure 2.4: Illustration of the streaming model: D is the dataset, $\{D_1, D_2, \ldots, D_T\}$ is a partition of D (i.e., $D = \uplus_{i=1}^{T} D_i$, where $\uplus$ denotes the disjoint union), and D^* is the solution produced by the algorithm.

2.1.3 Geometric data summarization

Apart from statistical and submodular approaches, summarization techniques originating from computational geometry have also received attention lately. **Geometric data summarization** has become a vital tool to tackle problems that lie at the intersection between geometry and Big Data. Among geometric summaries, **coresets** and **sketches** stand out.

2.1.3.1 Coresets

A **coreset** is a small representation of a dataset used to perform fast approximate inference with strong theoretical guarantees [Phillips(2017)]. Coresets are designed so that the result produced when running mining algorithms on such summaries closely resembles the outcome obtained when ran on the full dataset (see figure 2.5). This way, knowledge discovery tasks can be carried out orders of magnitude faster without provably sacrificing accuracy.

Formally, given a dataset D, a coreset is a weighted subset of D or the space where D resides. That is, fixed $D \subseteq \mathcal{U}$ for some finite universe $\mathcal{U}$, a coreset is a subset $D^* \subseteq D$ or a subset $D^* \subseteq \mathcal{U}$, where each datapoint $d \in D^*$ gets assigned a weight $w(d) \in \mathbb{R}$. A good example where both variants arise naturally stems from the clustering framework. If $k-$medoids and k-means are executed on a dataset $D \subseteq \mathcal{U}$, the former will return a coreset $D^*_{\text{medoids}} \subseteq D$, while the latter will yield a coreset $D^*_{\text{means}} \subseteq \mathcal{U}$, both with unitary weights (i.e, $w(d) = 1$, $\forall\, d \in D^*_{\text{means}} \cup D^*_{\text{medoids}}$).

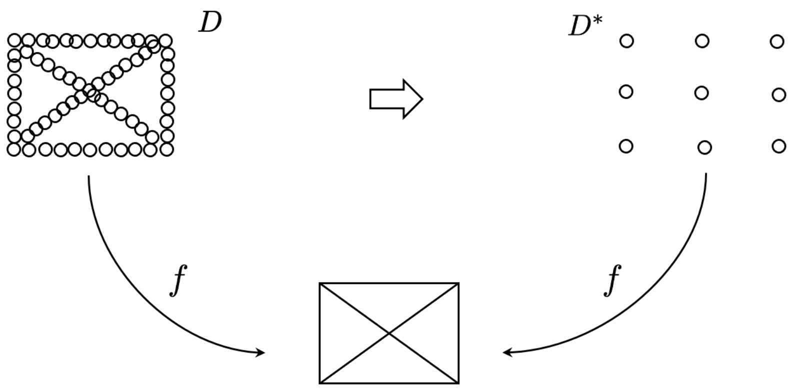


Figure 2.5: Illustrative example to demonstrate the use of coresets extracted from [Feldman(2020)]. D is the full dataset, D^* is a coreset of D, and f is a function (algorithm) that receives a set of points and outputs its clustering into linear segments. Running f on the coreset D^* should ideally produce the same result as that yield when ran on D, but faster.

2.1.3.2 Sketching

Sketching allows summarization of streaming data on the fly. A **sketch** is a brief synopsis of a dataset $D \subseteq \mathcal{U}$ that gets updated as new instances are received. More formally, a sketch is the image of D via a function f whose co-domain is a compact, easily updatable data structure.

Sketches are designed so that the update caused by each new piece of data is largely independent of the current state of the sketch. Despite this favorable behaviour, each sketch family only allows answering very limited questions about the data. Therefore, prior to the summarization exercise, the user must specify the type of queries the summary should be able to answer, as this will utterly determine the employed sketching technique.

Sketching techniques include: **Bloom filters**, **Count-Min sketching**, and **HyperLogLog sketching** [Cormode et al.(2011), Cormode(2017)]. Next, a brief description of each one is provided for completeness.

Bloom filters A Bloom filter is a binary string B of length $m << n = |D|$, along with k hash functions $h_1, \dots, h_k : \mathcal{U} \to \{1, \dots, m\}$ that independently map each element of the universe $\mathcal{U}$ to an index in B. Initially, B is the all zero string, but upon receiving $D \subseteq \mathcal{U}$, entry $B[h_j(d)]$ is set to 1 for each $d \in D$ and $1 \leq j \leq k$.

Bloom filters are designed to precisely answer membership queries. That is, given a datapoint $d \in \mathcal{U}$ they allow to efficiently find out whether $d \in D$ using a simple decision process: If $\exists\ j \in \{1, \dots, k\}$ such that $B[h_j(d)] = 0$, then $d \notin D$, and otherwise $d \in D$. The sketch ensures no false negatives, but it may report some false positives. Nonetheless, with a careful choice of k (namely, $k = \frac{m}{n} \cdot \ln 2$), the probability of the latter is minimized.

Count-Min sketches A Count-Min sketch is an array C of dimension $w \times m$, together with w hash functions $h_1, \ldots, h_w : \mathcal{U} \to \{1, \ldots, m\}$ (one for each row) that map each element of the universe $\mathcal{U}$ to an index inside their assigned row. Initially, C is the all zero matrix, but upon receiving a multi-set $D \subseteq \mathcal{U}$, each entry $D[i, j]$ is set to the number of elements in D mapped to position j by function h_i.

Count-Min sketches are designed to precisely answer frequency queries. Given a datapoint $d \in \mathcal{U}$ the sketch estimates the number of occurrences of d within D as $\min_{1 \le i \le w}\{h_i(d)\}$. Due to the nature of hash functions, the frequency count of any datapoint may be overestimated. However, upon choosing $m = 2/\epsilon$ and $w = \log \frac{1}{\delta}$, the estimate produced by the summary will have at most an error of $\epsilon \cdot \sum_{d \in \mathcal{U}} \text{freq}(d)$ with probability at least $1 - \delta$, where $\text{freq}(d)$ is the true frequency of d in D.

HyperLogLog sketches A HyperLogLog sketch is an array H of length m that holds numeric counters, initially set to 0. HyperLogLog sketches are designed to accurately answer cardinality queries. That is, they allow to approximately count the number of distinct elements in a multi-set $D \subseteq \mathcal{U}$. Every element in D is hashed to a binary string of fixed length, and subsequently hashed again into one of H's m buckets. Then, counter $H[i]$ becomes 2 to the power of the longest sequence of consecutive zeros found in any binary string inside bucket i. Since a sequence of k consecutive zeros will occur on average once in every 2^k entries, each counter holds an estimate of the number of distinct elements in D. Last, the harmonic mean of the counters is returned as the final estimate.

2.2 Data forgetting

All summarization techniques discussed up to this point share a similar philosophy when addressing data reduction. Namely, every datapoint in the input dataset D is treated as a first class citizen. Since the ultimate goal of summarization is building a synopsis that's as representative as possible of the complete dataset, summarization routines assume that every single record in D conveys a certain degree of valuable information. However, it's natural to wonder whether such an assumption is always reasonable. If a part of the dataset is never used / queried, will it still be beneficial to summarize it? Even more, will it still be worth keeping in any form?

Data forgetting provides a satisfactory answer to such questions by taking a different perspective towards data reduction. Forgetting methods acknowledge that value is not equally distributed among all regions in the given dataset. That is, upon fixing a notion of **value / importance**, there exist some portions in D that contain higher value than others. Hence, such routines seek identifying the dataset's relevant regions and "forgetting" everything that lies outside of them. Following, we provide an overview of existing forgetting techniques, each of which, given a datset D aim to exclusively retain a valuable subset of it.

2.2.1 Amnesia algorithms - Probabilistic data forgetting

Amnesia algorithms created a paradigm shift in the conception of the role of databases. Amnesia-scheme equipped databases have the power to controllably forget data entries. Hence, unlike traditional ones, they are not conceived as static objects whose sole purpose is storing data indefinitely, but as dynamic entities able to dispose of datapoints that won't be useful in the future [Kersten and Sidirourgos(2017)].

The amnesia model is very simple: Data arrives at successive time steps in form of equally-sized batches, and only B datapoints are held inside the database at any point in time. That is, at each time step t, m new records arrive and m extisting entries in the database are forgotten to meet the storage budget B. The way such removal is executed depends on the specific amnesia strategy. In general, the m least useful datapoints are deleted at each step t.

It is important to note that usefulness is an extremely domain-dependant concept. Depending on the type of data and case application, it can vary unimaginably. Three notions of data usefulness have been proposed, each of which result in a different amnesia scheme.

2.2.1.1 Temporal based amnesia algorithms

Temporal based amnesia schemes understand usefulness as a temporal dimension. The highest utility entries in the database are either recently added ones (**retrograde amnesia**) or anciently added ones (**anterograde amnesia**). At each time step, retrograde (resp. anterograde) based amnesia schemes randomly remove items proportionally (resp. inverse-proportionally) to the amount of time they have been part of the database. That is, in the case of retrograde amnesia, newer entries have lower probability of being forgotten than older ones; while for anterograde amnesia, the opposite holds.

Examples of amnesia routines include **FIFO-amnesia**, **uniform-amnesia**, and **ante-amnesia** algorithms. The first boils down to a (deterministic) temporal sliding-window. At each time step, the (retrograde) FIFO-amnesia algorithm stores the m newly received records and evicts the m oldest ones, exclusively remembering the latest entries inserted into the databse. For the contrary, the second is able to retain datapoints spread over a larger segment of the time line by using a randomized procedure. At each time step, the (retrograde) uniform-amnesia algorithm stores the m newly received records and forgets m other entries uniformly at random. Hence, at any step, all datapoints have the same probability of being forgotten, but older ones have been candidates multiple times. Last, at each time step t, the (anterograde) ante-amnesia algorithm stores the m newly received datapoints, and forgets m entries uniformly at random among the latest stored records including those received at time t.

2.2.1.2 Query based amnesia algorithms

Query based amnesia schemes conceive usefulness as frequency of retrieval. The highest utility datapoints are the most requested ones according to a query-log Q. Every entry in the database is extended with a frequency counter that indicates the number of times it has been accessed by a query in Q. At each time step, the m newly received records get stored in the database with an assigned frequency count of zero, while m other datapoints are randomly removed inverse-proportionally to their frequency counter. That is, entries with a higher frequency of retrieval (i.e., those that have been requested by queries up to that point the most) have lower probability of being forgotten than those with a lower one (i.e., those that have not been asked so often up to that point).

It is worth remarking that the query-log Q dynamically increases over time, and consequently, frequency counters change between time steps. That is, all queries taken place between time steps t and $t + 1$ cause updates on the counters of the B datapoints present in the database between such times. Hence, a point that had a low frequency counter at time t may have a very high counter at step $t + 1$ if many queries showed interest in the former during that portion of the timeline. Consequently, the most historically requested datapoints tend to be the ones retained by query based amnesia schemes.

2.2.1.3 Spatial based amnesia algorithms

Spatial based amnesia algorithms perceive usefulness as freshness. Under this interpretation, non-"infected with mold" datapoints are those with the highest utility. Every newly entry in the database is assigned a freshness value of 1 that decreases over time upon infection or direct contact with an infected entry (much like mold spots spread on cheese). At each time step, **spatial based amnesia** schemes assign a freshness score of 1 to the m newly received records, randomly infect a certain number of records in the database, and decrease the freshness score of every infected entry as well as that of all its neighbouring records. After this process, the m datapoints with the lowest freshness value get forgotten. Alternatively, if there are no budget concerns, datapoints can be simply forgotten when their freshness reaches 0.

Freshness is tightly related with **rotting** [Kersten(2015), Kersten(2016)]. **Data rotting** is inspired in nature's own rotting process: over time, objects rot away; hence, so should data. In fact, spatial based amnesia schemes essentially rot the database away over time.

2.2.2 Data valuation - Score based data forgetting

A deterministic alternative to probability-reliant amnesia algorithms are **data valuation** routines. Given a dataset D, each datapoint $d \in D$ is assigned a numerical score, and only the most valuable instances are retained. That is, fixed a threshold $\theta \in \mathbb{R}$ (resp. a budget $B \in \mathbb{N}$), only the entries whose value exceed θ (resp. the top B-highest-scoring records) are kept.

The quality of a datapoint can be assessed based on a number of metrics including its **accessibility**, **age**, **completeness**, **consistency**, **currency**, **heterogeneity**, **interpretability**, **objectivity**, **provenance**, **relevancy**, and **uniqueness** [Wang and Strong(1996)]. A function $f : D \mapsto \mathbb{R}$ that maps each datapoint $d \in D$ to a numerical score $f(d) \in \mathbb{R}$ representing its value within D, can be easily constructed by exploiting such quality metrics.

2.2.3 Submodular maximization - Optimization based data forgetting

Apart form data valuation, the latest direction in (deterministic) data forgetting has been casting the problem as an optimization-based subset selection exercise. That is, given a dataset D and some measure of subset utility $f : \mathcal{P}(D) \to \mathbb{R}$, retaining a subset $D^* \subseteq D$ that attains the highest utility according to f. Two recent works steaming from the field of e-commerce, exploit submodular function maximization for data forgetting in the context of graph and image data.

2.2.3.1 Graph data

Given a dataset D, a weighted directed graph $G = (V = D, E, W_V, W_E)$, a budget B, and a utility function $f : \mathcal{P}(D) \to [0, 1]$, a solution to the **preference cover problem** [Gershtein et al.(2020)] is a subset $D^* \subseteq D$ of size B with maximal utility. That is,

$$D^* = \underset{D' \subseteq D,\, |D'| = B}{\arg\max} \quad f(D').$$

Each vertex $v \in V$ corresponds to an item in D, and its weight, $W_V(v) \in [0, 1]$, indicates the probability of v being requested by a costumer. Moreover, the set of neighbours of v, $R_v(D) := \{u \,|\, (v, u) \in E, u \in D\}$, comprises those items in D considered as feasible alternatives to v. Directed edge $(v, u) \in E$ indicates that a consumer is willing to buy, with probability $W_E(v, u)$, item u as an alternative to v in case the latter is missing. Given a retained subset $D' \subseteq D$ and a costumer request for $v \in D$, the request is meet if $v \in D'$. Otherwise, the request may be meet by a neighbour $u \in R_v(D')$ with probability of $W_E(v, u)$. Utility function $f : \mathcal{P}(D) \mapsto [0, 1]$ maps each subset $D' \subseteq D$ to the probability that a request drawn from the distribution indicated by the node weights is satisfied. Naturally, the goal is selecting a subset D^* that maximizes such probability. An illustrative example is shown in figure 2.6.

The preference cover problem comes in two flavours: **independent** and **normalized**. The independent variant assumes independence between all edge weights. That is, for a retained subset $D' \subseteq D$ and a requested item $v \notin D'$, the probability that a neighbour $u_1 \in R_v(D')$ matches the request is not affected by whether or not a different neighbour $u_2 \in R_v(D')$ matches it. For the contrary, the normalized variant assumes dependence between edge weights, and thus, $\sum_{u \in R_v(D)} W_E(v, u) = 1, \; \forall v \in V$. Each variant comes with a different instantiation of f. Namely,

$$f_{\text{independent}}(D') = \sum_{v \in D'} W_V(v) + \sum_{v \in V \setminus D'} W_V(v) \cdot \left(1 - \prod_{u \in R_v(D')} (1 - W_E(v, u)) \right),$$

$$f_{\text{normalized}}(D') = \sum_{v \in D'} W_V(v) + \sum_{v \in V \setminus D'} W_V(v) \cdot \left(\sum_{u \in R_v(D')} W_E(v, u)) \right).$$

Since $f_{\text{independent}}$ is non-negative, monotone, and submodular, the `GREEDY` algorithm (algorithm 1) is resorted on to address the independent variant of the problem. Sadly, $f_{\text{normalized}}$ does not exhibit submodularity. Nonetheless, this version of the problem is equivalent to **vertex cover**, and consequently, `GREEDY` still provides a good approximation factor.

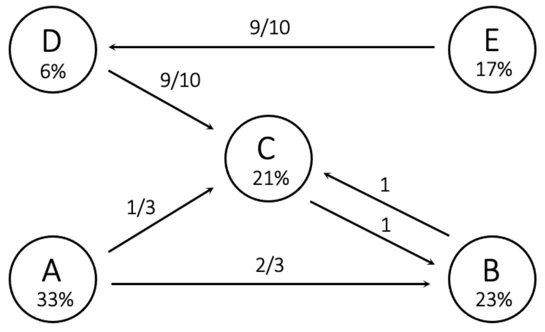


Figure 2.6: Toy example extracted from [Gershtein et al.(2020)]. The full dataset is $\{A, B, C, D, E\}$ and the budget is $B = 2$. The most popular item is A, appearing in 33% of the requests. The least popular item is D, appearing in 6% of the requests. Items C and B are perfect substitutes, item D is an almost perfect substitute for E, and C is an almost perfect substitute of D. Upon fixing $f = f_{\text{normalized}}$ (formula provided below), $D^* = \{B, D\}$.

2.2.3.2 Image data

Given an image dataset D, a budget B, and a set of photo albums $Q \subseteq \mathcal{P}(D)$, a solution to the **photo archive reduction problem** (PAR) [Davidson et al.(2022)] is a subset

$$D^* = \underset{D' \subseteq D,\, C(D') \leq B}{\arg\max} \quad f_{\text{PAR}}(D')$$

where,

$$f_{\text{PAR}}(D') = \sum_{q \in Q} W(q) \cdot \sum_{p \in q} R(q,p) \cdot SIM(q,p,NN(q,p,D')).$$

Even though the objective seems very intricate, the individual components that conform it are quite simple. First, function $C : \mathcal{P}(D) \mapsto \mathbb{R}_+$ maps each subset $D' \subseteq D$ to its storage cost. Second, mapping $W : Q \mapsto \mathbb{R}_+$ assigns a positive weight to each album $q \in Q$ that reflects its importance. Third, function $R : Q \times D \mapsto [0,1]$ gives a normalized score to each image $d \in q$ that indicates how relevant it is for album $q \in Q$. Fourth, $SIM : Q \times D \times D \mapsto [0,1]$ produces a normalized measure of similarity between any given pair of images in the context of $q \in Q$. Last, $NN(q,d,D') = \arg\max_{d' \in D' \cap q(D)} SIM(q,d,d')$ is the nearest neighbour of d in D' in the context of q. As a consequence, the solution D^* contains the most relevant images for the most important albums.

Despite its intricacy, utility function $f_{\text{PAR}}(\cdot)$ is non-negative, monotone, and submodular. Hence, the PAR problem can be seen as an instance of the knapsack-constained submodular function maximization problem, for which, a cost-aware variant of the GREEDY algorithm (see algorithm 5) guarantees a $\frac{1-1/e}{2}$ approximation factor [Leskovec et al.(2007)].

Algorithm 5: COST-BENEFIT-GREEDY

```
Data: Dataset $D$, submodular monotone objective $f : \mathcal{P}(D) \mapsto \mathbb{R}_+$, cost function
      $C : \mathcal{P}(D) \mapsto \mathbb{R}_+$, and budget $B \in \mathbb{N}$.
Result: Subset $D^* \subseteq D$ such that $C(D^*) \leq B$ and $f(D^*) \geq \frac{1-1/e}{2} \cdot OPT$, where $OPT$ is
        the value of the optimal (intractable) solution.
$D_0 \leftarrow \emptyset, t \leftarrow 1$;
while $\exists\, d \in D \setminus D_{t-1} : C(D_{t-1} \cup \{d\}) \leq B$ do
    $d_t \leftarrow \arg\max_{d \in D;\, C(\{v\}) \leq B - C(D_{t-1})} [f(D_{t-1} \cup \{d\}) - f(D_{t-1})] / C(\{d\})$;
    $D_t \leftarrow D_{t-1} \cup \{d_t\}$;
    $D \leftarrow D \setminus \{d_t\}$;
    $t \leftarrow t + 1$;
end
$T \leftarrow t$;
return $D^* = \arg\max_A \{f(A) : A \in \{D_T, \texttt{GREEDY}(D)\}\}$;      /* Note that the
  COST-BENEFIT-GREEDY algorithm returns the best solution
  between the cost-aware solution and that produced by the
  cost-oblivious GREEDY algorithm (algorithm 1).  */
```

2.3 Summarization vs Forgetting, a bird's-eye view

As previously mentioned, summarization algorithms conceive every datapoint as equally valuable. Consequently, they often address the data reduction problem by carving out representative subsets of the input dataset guided by value-unaware submodular objectives. For the contrary, forgetting routines understand data-value as an unequally distributed feature within the dataset. Therefore, they seek identifying such worthy regions by embedding the notion of data-importance as an essential ingredient in the algorithmic exercise. A hierarchical schema of all the summarization and forgetting techniques discussed in the chapter is illustrated in figure 2.7.

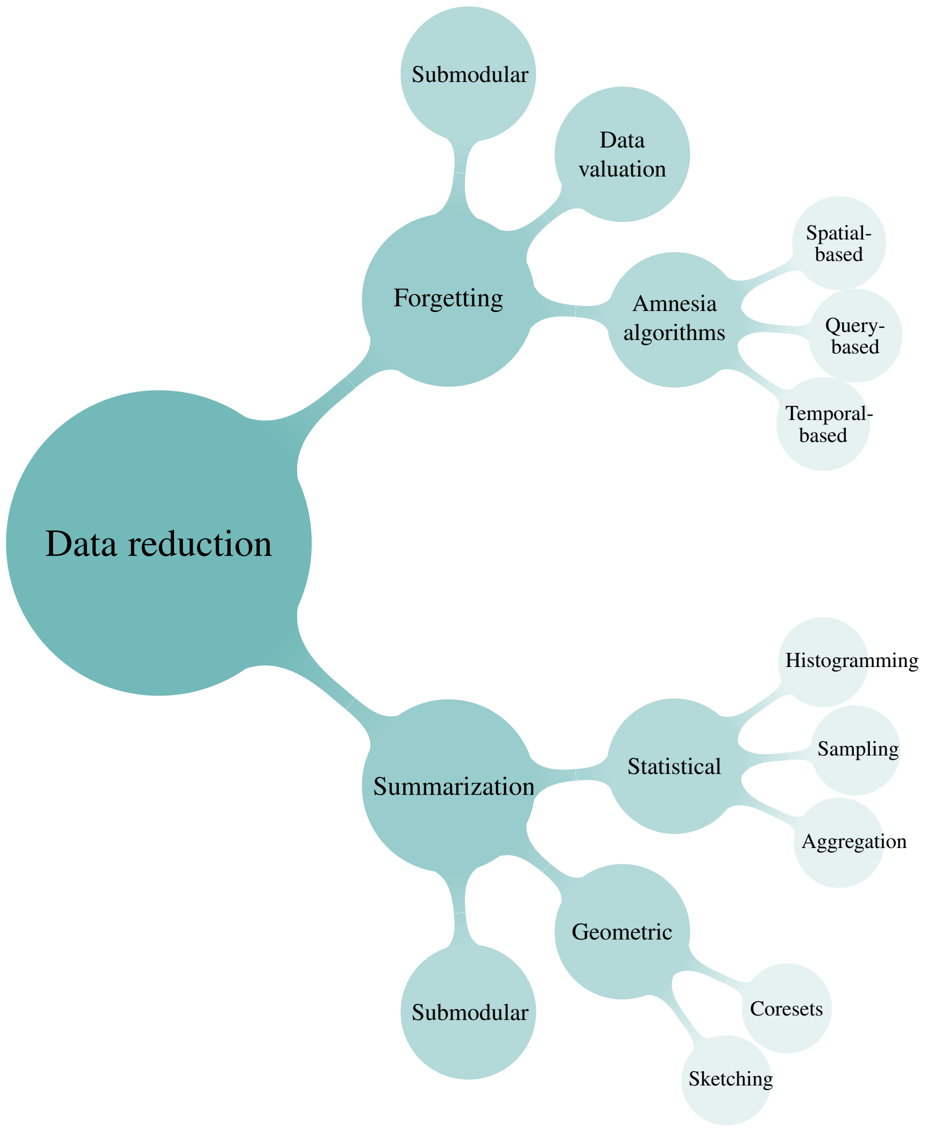


Figure 2.7: Landscape of existing data reduction techniques

Chapter 3

Stochastic submodular data forgetting

3.1 Preliminaries

3.1.1 Relational datasets and conjunctive queries

Let $\mathcal{A} = \{A_1, \dots, A_m\}$ be a collection of attributes, each of which is associated with a finite domain Dom_{A_i}. A **tuple** is an $m-$dimensional vector $d = (v_1, \dots, v_m)$ where $v_i \in Dom_{A_i}$. Equivalently, d can be expressed as a conjunction of conditions $A_i = v_i$, for $1 \leq i \leq m$. That is,

$$d = \bigwedge_{i=1}^{m} A_i = v_i.$$

For any tuple d, $\mathfrak{Constrs}(d) = \{A_1 = v_1, \dots, A_m = v_m\}$ denotes the set of conditions of d. The set of all possible tuples $\mathcal{U} = Dom_{A_1} \times \cdots \times Dom_{A_m}$ constitutes the **universe**. A **dataset** is a set $D \subseteq \mathcal{U}$ (i.e., a finite collection of tuples).

A **query** q is a conjunction of atomic conditions of the form $A_i = v_i$, where $A_i \in \mathcal{A}$ and $v_i \in Dom_{A_i}$. That is,

$$q = \bigwedge_{i \in I} A_i = v_i,$$

where $A_i \in \mathcal{A}$, $v_i \in Dom_{A_i}$, and $I \subseteq \{1, \dots, m\}$. For any query q, $\mathfrak{Constrs}(q) = \{A_i = v_i : i \in I\}$ denotes the set of conditions of q. A tuple $d \in \mathcal{U}$ is said to **satisfy** a query q if $\mathfrak{Constrs}(q) \subseteq \mathfrak{Constrs}(d)$. The **universe of a query** q, denoted as $\mathcal{U}_q$, is the set of all the tuples in the universe $\mathcal{U}$ that satisfy the query q. The **answer set of a query** q on a dataset D, denoted as $q(D)$, is the set of tuples in D that satisfy query q. Furthermore, given a query q and datasets $D' \subseteq D \subseteq \mathcal{U}$, the **precision** of q in D' with respect to D, is the fraction of tuples shared by answer sets $q(D')$ and $q(D)$. Formally,

$$Precision(q, D', D) = \frac{|q(D')|}{|q(D)|} \in [0, 1].$$

Alternatively, a query q can be conceived as a function that maps any given dataset $D \subseteq \mathcal{U}$ to its unique answer set. That is,

$$\begin{aligned} q : \mathcal{P}(\mathcal{U}) &\to \mathcal{P}(\mathcal{U}_q), \\ D &\to q(D). \end{aligned}$$

Under this functional interpretation, queries are monotone and distribute over set union and intersection. In other words, the following expressions hold $\forall\, q$ and $\forall\, D_1' \subseteq D_1 \subseteq \mathcal{U}, D_2 \subseteq \mathcal{U}$:

$$q(D_1') \subseteq q(D_1), \quad q(D_1 \cup D_2) = q(D_1) \cup q(D_2), \quad q(D_1 \cap D_2) = q(D_1) \cap q(D_2).$$

3.1.2 Set similarity functions

Set similarity functions play a central role in **data forgetting**, as they allow measuring the extent to which answer sets $q(D')$ and $q(D)$ are alike for any query q and datasets $D' \subseteq D \subseteq \mathcal{U}$. Formally,

Definition 3.1.1. *Given a dataset $D \subseteq \mathcal{U}$, a **set similarity function** is a mapping $S : \mathcal{P}(D) \times \mathcal{P}(D) \mapsto \mathbb{R}$ such that $\forall\, A, B, C \subseteq D$:*

(1) $0 \leq S(A, B) \leq 1$ ***(Normalized non-negativity)**,*

(2) $S(A, A) = 1$ ***(Self-similarity)**,*

(3) $S(A, B) = S(B, A)$ ***(Symmetry)**.*

■

Arguably, the most famous set similarity function is the **Jaccard** similarity [Leskovec et al.(2020)]. Given a dataset D and subsets $A, B \subseteq D$, the Jaccard similarity between sets A and B is defined as

$$Jaccard(A, B) := \frac{|A \cap B|}{|A \cup B|}.$$

3.2 Data forgetting

Data forgetting is a novel approach towards big data reduction. Given a massive dataset D, the forgetting paradigm acknowledges that value is not equally distributed among all regions of D. That is, upon fixing a notion of data-value/importance, there exist some portions in the dataset that are more valuable than others. **Data forgetting algorithms** seek reducing D by identifying its relevant regions and **forgetting** (i.e., deleting) everything that lies outside of them.

Clearly, there is no single best definition for data-relevance. Nonetheless, a reasonable way of assigning value to data is by considering the expected workload of queries. This way, the most valuable regions of a given dataset D are those that allow answering the expected query workload in the most faithful manner. We formalize this intuitive notion of data-importance by means of the objective (objective (3.1)) of an algorithmic subset selection exercise (problem 1).

Problem 1 (Data forgetting). *Given:*

- *A dataset $D = \{d_1, \dots, d_n\}$;*
- *A set of queries Q together with a probability distribution $\mathcal{Q}$ defined on Q;*
- *A set similarity function $S : \mathcal{P}(D) \times \mathcal{P}(D) \mapsto \mathbb{R}_+$;*
- *A budget $B \in \mathbb{N}$;*

find a subset $D^* \subseteq D$ such that

$$D^* = \underset{D' \subseteq D,\ |D'| \leq B}{\arg\max} \quad \mathbb{E}_{q \sim \mathcal{Q}}[f_q(D')], \tag{3.1}$$

where $f_q(D') = S(q(D), q(D'))$.

Solution D^* is a subset of D composed of at most B tuples for which the most likely queries $q \in Q$ according to distribution $\mathcal{Q}$, have answer sets $q(D^*)$ of maximal resemblance to $q(D)$ under set similarity function S. That is, D^* comprises those (no more than B) tuples in D that best allow coping with the expected query workload; aligning with our desired notion of data-importance.

Both distribution $\mathcal{Q}$ and similarity function S are crucial ingredients that utterly determine D^*. On the one hand, distribution $\mathcal{Q}$ reflects the relative frequency of usage of each query $q \in Q$, and consequently, usually corresponds to the empirical distribution built from a query-log with records in Q. On the other hand, similarity function S quantifies the discrepancy between the query results achieved on the retained dataset D^* and those obtained on the full dataset D (i.e., between answer sets $q(D^*)$ and $q(D)$).

3.2.1 Independent data forgetting

The simplest version of the problem arises upon assuming pairwise disjointness between the content of tuples in the input dataset D. That is, upon assuming that the information residing in any $d \in D$ cannot be totally/partially found within other tuples in D. In this scenario, fixed $q \in Q$, since the topology of D is ignored, every datapoint $d \in q(D)$ is of equal importance to the answer set. For this reason, given $D' \subseteq D$, the most natural way of measuring the discrepancy between answer sets $q(D')$ and $q(D)$ is by looking at the fraction of tuples that both share. In other words, via the precision of query q in D' with respect to D.

Given that query precision is the most natural way of measuring the discrepancy between answer sets in the independent setting, it remains checking whether $Precison(q, D', D)$ can be written as $S(q(D), q(D'))$, for a set similarity function S. The following proposition provides a satisfactory answer and unveils a beautiful connection between query precision and the Jaccard similarity.

Proposition 3.2.1. *Given a dataset D and a query q,*

$$Precison(q, D', D) = Jaccard(q(D), q(D')).$$

Proof. The result essentially follows from the monotonicity of conjunctive queries when understood as mappings (i.e., the fact that $\forall\, q$, and $D' \subseteq D \implies q(D') \subseteq q(D)$).

$$Precision(q, D', D) = \frac{|q(D')|}{|q(D)|} = \frac{|q(D') \cap q(D)|}{|q(D') \cup q(D)|} = Jaccard(q(D), q(D')).$$

□

We refer to problem 1 under the Jaccard similarity as the **independent data forgetting problem**. Interestingly, as we will soon prove, the former is an instance of the classic **0-1 Knapsack problem**; and consequently, it's in weak-NP and it has an exact solution that can be found in pseudo-polynomial time.

Problem 2 (0-1 Knapsack). *Given a set of n items numbered from 1 to n, each with an assigned weight w_i and value v_i, along with a maximum weight capacity W; maximize*

$$\sum_{i=1}^{n} v_i \cdot x_i, \tag{3.2}$$

subject to

$$\begin{cases} \sum_{i=1}^{n} w_i \cdot x_i \leq W, \\ x_i \in \{0, 1\}. \end{cases}$$

In order to show that independent data forgetting is an instance of 0-1 Knapsack, we rely on the following lemma concerning answer set cardinality decomposition.

Lemma 3.2.2. *For any given datasets $D' \subseteq D$ and query q,*

$$|q(D')| = \sum_{d \in D'} |q(\{d\})|.$$

Proof.

$$\begin{aligned} D' = \uplus_{d \in D'}\{d\} &\underset{(1.1)}{\implies} q(D') = q(\uplus_{d \in D'}\{d\}) \\ &\underset{(1.2)}{\implies} q(D') = \uplus_{d \in D'} q(\{d\}) \\ &\underset{(1.3)}{\implies} |q(D')| = |\uplus_{d \in D'}\, q(\{d\})| \\ &\underset{(1.4)}{\implies} |q(D')| = \sum_{d \in D'} |q(\{d\})|. \end{aligned}$$

(Note: $\uplus$ denotes the disjoint union).

(1.1) holds because the answer set of any query q on any subset $D' \subseteq D$ is unique.

(1.2) holds because when q is understood as a mapping it distributes under set union.

(1.3) holds because equal sets have the same cardinality.

(1.4) holds because of the inclusion-exclusion principle in the case of a disjoint union. □

It's worth noting that the above result has a very intuitive combinatorial interpretation: Fixed any query q and datasets $D' \subseteq D$, the cardinality of answer set $q(D')$ can be computed by sequentially checking whether each datapoint $d \in D'$ belongs to $q(D)$. Next, we proceed to the main result of the section where the aforementioned result plays a central role.

Theorem 3.2.3. *Independent data forgetting is an instance of 0-1 Knapsack.*

Proof. If $S(q(D), q(D')) = Jaccard(q(D), q(D')) = Precision(q, D', D)$, objective (3.1) becomes finding a subset $D^* \subseteq D$ such that

$$\begin{aligned}
D^* = &\underset{D' \subseteq D,\, |D'| \leq B}{\arg\max} \quad \mathbb{E}_{q \sim \mathcal{Q}}\left[\frac{|q(D')|}{|q(D)|}\right] \\
= &\underset{D' \subseteq D,\, |D'| \leq B}{\arg\max} \quad \sum_{q \in Q} \mathbb{P}_{\mathcal{Q}}(q) \cdot \frac{|q(D')|}{|q(D)|} \\
\underset{\text{Lemma 3.2.2}}{=} &\underset{D' \subseteq D,\, |D'| \leq B}{\arg\max} \quad \sum_{q \in Q} \mathbb{P}_{\mathcal{Q}}(q) \cdot \frac{\sum_{d \in D'} |q(\{d\})|}{|q(D)|} \\
= &\underset{D' \subseteq D,\, |D'| \leq B}{\arg\max} \quad \sum_{q \in Q} \mathbb{P}_{\mathcal{Q}}(q) \cdot \sum_{d \in D'} \frac{|q(\{d\})|}{|q(D)|} \\
= &\underset{D' \subseteq D,\, |D'| \leq B}{\arg\max} \quad \sum_{q \in Q} \sum_{d \in D'} \mathbb{P}_{\mathcal{Q}}(q) \cdot \frac{|q(\{d\})|}{|q(D)|} \\
= &\underset{D' \subseteq D,\, |D'| \leq B}{\arg\max} \quad \sum_{d \in D'} \sum_{q \in Q} \mathbb{P}_{\mathcal{Q}}(q) \cdot \frac{|q(\{d\})|}{|q(D)|} \\
= &\underset{D' \subseteq D,\, \sum_{d \in D'} 1 \leq B}{\arg\max} \quad \sum_{d \in D'} \mathbb{E}_{q \sim \mathcal{Q}}\left[\frac{|q(\{d\})|}{|q(D)|}\right].
\end{aligned}$$

Hence, the independent version of the data forgetting problem is an instance of the 0-1 Knapsack problem where the set of n items corresponds to the dataset D, $v_d = \mathbb{E}_{q \sim \mathcal{Q}}\left[\frac{|q(\{d\})|}{|q(D)|}\right]$, $w_d = 1$, and $W = B$.

It's worth mentioning that this result extends beyond the simple cardinality constraint. In particular, it holds for the (more general) Knapsack constraint. If the cost of each tuple $d \in D$ is given by a cost function $C : D \to \mathbb{R}_+$ where $C(D') := \sum_{d \in D'} C(d)\ \forall D' \subseteq D$, the problem reduces to the 0-1 Knapsack problem where the set of n items corresponds to the dataset D, $v_d = \mathbb{E}_{q \sim \mathcal{Q}}\left[\frac{|q(\{d\})|}{|q(D)|}\right]$, $w_d = C(d)$, and $W = B$. □

Despite being exactly solvable in pseudo-polynomial time, the independent model has a drawback: Since pairwise disjointness between tuples' content is assumed, the inner relationships between tuples in D are ignored. As we shall see next, a (dependent) version of the data forgetting problem that takes into account the topology of D, follows naturally under basic assumptions.

3.2.2 Dependent data forgetting

Toy example 1.0.1 exhibits a somewhat general principle: If a requested tuple d is not retained (i.e., $d \in q(D)$ but $d \notin D'$), its information may be totally/partially encompassed by another retained tuple d' that's similar to the former (i.e., one such that $d' \in D' \cap q(D)$). This topology-aware notion of similarity between answer sets can be formalized via mapping

$$S(q(D), q(D')) = \frac{1}{|q(D)|} \cdot \sum_{d \in q(D)} \max_{d' \in q(D')} \text{sim}(d, d'), \tag{3.3}$$

where $\text{sim}(\cdot,\cdot)$[1] indicates the degree of similarity between any given pair of data points. That is, the similarity between answer sets $q(D)$ and $q(D')$ is the sum of similarities between each element in $q(D)$ and its closest neighbour inside $q(D')$, according to $\text{sim}(\cdot,\cdot)$. Generally, if D is categorical, it's natural to consider

$$\text{sim}(d, d') = Jaccard(d, d') := \frac{|\mathfrak{Constrs}(d) \cap \mathfrak{Constrs}(d')|}{|\mathfrak{Constrs}(d) \cup \mathfrak{Constrs}(d')|}, \tag{3.4}$$

(see figure 1.1). For the contrary, if D is numerical, it's reasonable to choose

$$\text{sim}(d, d') = cos(d, d') := \frac{d \cdot d'}{||d|| \cdot ||d'||}. \tag{3.5}$$

Furthermore, in the particular case where

$$\text{sim}(d, d') = \begin{cases} 1 & \text{if } d = d', \\ 0 & \text{if } d \neq d', \end{cases}$$

(3.3) reduces to the independent variant's objective.

We refer to problem 1 under function (3.3) as the **dependent data forgetting problem**. Not surprisingly, taking into account the inner relationships between tuples in D makes the algorithmic task significantly harder. In fact, the problem cannot longer be mapped into an instance of the 0-1 Knapsack problem, and function (3.3) can't be identified with any proper set similarity function (see appendix A.1). Fortunately, dependent data forgetting can be seen as an instance of the **discrete stochastic submodular maximization problem**, recently introduced in [Karimi et al.(2017), Hassani et al.(2017)].

Problem 3 (Discrete stochastic submodular maximization). *Given a ground set of items V and a probability distribution $\mathcal{D}$; find a subset $S^* \subseteq V$ such that*

$$S^* = \arg\max_{S' \in \mathcal{I}} \; f(S') := \arg\max_{S' \in \mathcal{I}} \; \mathbb{E}_{\theta \sim \mathcal{D}}[f_\theta(S')], \tag{3.6}$$

where functions $f_\theta : \mathcal{P}(V) \to \mathbb{R}_+$ are non-negative, monotone, and submodular; and $\mathcal{I}$ is a general

[1] $\text{sim}(\cdot,\cdot)$satisfies the axioms of 3.1.1 at the data level

matroid constraint. *That is, $\mathcal{I} \subseteq \mathcal{P}(V)$ is a collection of subsets of V satisfying the following two properties:*

- $A \subseteq B \subseteq V$ *and* $B \in \mathcal{I} \implies A \in \mathcal{I}$,
- $A, B \in \mathcal{I}$ *and* $|B| > |A| \implies \exists e \in B \setminus A$ *such that* $A \cup \{e\} \in \mathcal{I}$.

Collection $\mathcal{I} = \{D' \subseteq D \ : \ |D'| \leq B\} \subseteq \mathcal{P}(D)$ is a matroid for any dataset D and budget B. Furthermore, functions as in (3.3) are members of a normalized thresholded variant of the well known (non-negative, monotone, submodular) family of facility location data summarization objectives (introduced in section 2.1.2.1). In other words, for any query q, set function $f_q(D')$ as in (3.3) satisfies that

$$f_q(D') = \frac{1}{|q(D)|} \cdot f_{\text{facility location}}(q(D')).$$

Consequently,

Proposition 3.2.4. *The objective of the dependent data forgetting problem can be written as*

$$f(D') = \mathbb{E}_{(d,q)\sim\mathcal{D}}\left[f_{q,d}(D')\right],$$

where functions

$$f_{q,d}(D') = \max_{d' \in q(D')} sim(d, d'). \tag{3.7}$$

are non-negative, monotone and submodular; and $\mathcal{D}$ is the probability distribution with mass function $\mathbb{P}_{\mathcal{D}}(d, q) = \frac{|q(\{d\})|}{|q(D)|} \cdot \mathbb{P}_{\mathcal{Q}}(q)$.

Proof. See appendix A.2 and A.3. □

All in all, we have cast dependent data forgetting as an instance of the discrete stochastic submodular maximization problem. Even more, we have observed that the problem's objective is a stochastic variant of the facility location function, and thus doubly stochastic (proposition 3.2.4).

3.2.3 A solution to the dependent data forgetting problem

Naturally, due to the aforementioned connection, existing stochastic submodular maximization technology should be exploited to solve dependent data forgetting. For this reason, we have decided to lift the problem into the continuous domain via **multilinear continuation**; solve the continuous version of the problem using a continuous stochastic submodular maximization procedure called **Stochastic Continuous Greedy** (SCG); and map the continuous solution back into the discrete domain via **randomized pipage rounding**. This process yields a solution with $(1 - 1/e - \epsilon)$ approximation guarantee, in expectation, to the optimal (intractable) solution. A bird's eye view of the solution's workflow is depicted in figure 3.1. The technical details will be provided in the following.

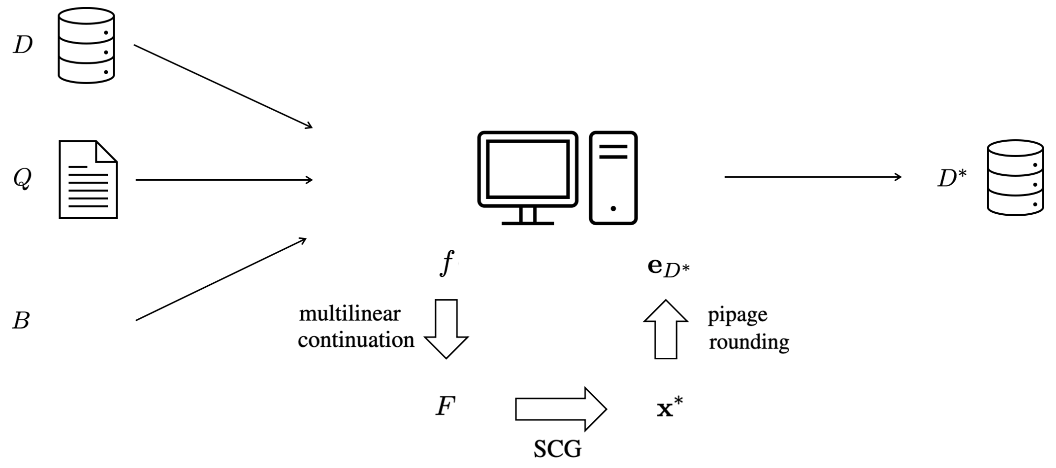


Figure 3.1: Workflow of the solution to the dependent data forgetting problem. **Input**: dataset D, query-log Q, and budget B. Query-log Q is pre-processed to extract $(Q, \mathcal{Q})$, where Q refers to the unique queries in the query-log (abusing notation), and $\mathcal{Q}$ to the empirical distribution. The objective is $f(D') = \mathbb{E}_{(d,q)\sim\mathcal{D}}\, f_{q,d}(D')$ with $f_{q,d}(D')$ as in (3.7), and $F(\mathbf{x})$ its multilinear extension. Vector $\mathbf{x}^*$ refers to the outcome of the SCG algorithm; and $\mathbf{e}_{D^*}$ that of the pipage rounding scheme. **Output**: Subset $D^* \subseteq D$ containing element $d_i \in D$ if the i-th coordinate of $\mathbf{e}_{D^*}$ takes the value 1. The retained subset D^* is such that $f(D^*) \geq (1 - 1/e - \epsilon) \cdot OPT$, in expectation, where OPT denotes the optimal (intractable) solution of the dependent data forgetting problem.

3.2.3.1 Multilinear continuation

Multilinear continuation is a technique that allows to continuously extend any set function while preserving its monotonicity and submodularity. More precisely, given a dataset $D = \{d_1, \dots, d_n\} \subseteq \mathcal{U}$, and a set function $f : \mathcal{P}(D) \to \mathbb{R}$, by identifying subsets $D' \subseteq D$ with binary vectors $\mathbf{e}_{D'} \in \mathbb{R}^n$ in which the i-th component takes value 1 if tuple $d_i \in D'$ and 0 otherwise, f can be re-written as a function $\tilde{f}$ defined over the corners of the unit cube in $\mathbb{R}^n$, $n = |D|$. Namely, $\tilde{f} : [0,1]^n \to \mathbb{R}$ such that $\tilde{f}(\mathbf{e}_{D'}) = f(D')$. The **multilinear extension** of f [Vondrák(2008)], denoted F, extends $\tilde{f}$ to the entire unit cube $[0,1]^n \subseteq \mathbb{R}^n$ as follows: $\forall\, \mathbf{x} = (x_1, \dots, x_n) \in [0,1]^n$,

$$F(\mathbf{x}) = \sum_{D' \subseteq D} f(D') \prod_{d_i \in D'} x_i \prod_{d_j \notin D'} (1 - x_j).$$

Multilinear continuation preserves monotonicity and submodularity. That is, for any monotone submodular set function f, its multilinear extension F is (continuous) submodular, (continuous) monotone, and DR-submodular. Formally, a continuous function $F : \mathcal{X} \subseteq \mathbb{R}^n_+ \to \mathbb{R}_+$ is said to be (continuous) **submodular** if $\forall\, \mathbf{x}, \mathbf{y} \in \mathcal{X}$,

$$F(\mathbf{x}) + F(\mathbf{y}) \geq F(\mathbf{x} \vee \mathbf{y}) + F(\mathbf{x} \wedge \mathbf{y}),$$

where $\mathbf{x} \vee \mathbf{y} := max(\mathbf{x}, \mathbf{y})$ (component wise) and $\mathbf{x} \wedge \mathbf{y} := min(\mathbf{x}, \mathbf{y})$ (component wise). Moreover, a (continuous) submodular function F is (continuous) **monotone** if $\forall\ \mathbf{x}, \mathbf{y} \in \mathcal{X}$,

$$\mathbf{x} \leq \mathbf{y} \implies F(\mathbf{x}) \leq F(\mathbf{y}).$$

Furthermore, a differentiable submodular function F is called **DR-submodular** if $\forall\ \mathbf{x}, \mathbf{y} \in \mathcal{X}$,

$$\mathbf{x} \leq \mathbf{y} \implies \nabla F(\mathbf{x}) \geq \nabla F(\mathbf{y}).$$

3.2.3.2 The Stochastic Continuous Greedy algorithm

The Stochastic Continuous Greedy (SCG) algorithm, recently introduced in [Mokhtari et al.(2018)], is a stochastic variant of gradient ascent originally designed to solve the **continuous stochastic submodular maximization** problem.

Problem 4 (Continuous stochastic submodular maximization). *Given a probability distribution $\mathcal{D}$; find a vector $\mathbf{x}^* \in \mathbb{R}_+^n$ such that*

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathcal{C}}{\arg\max}\ F(\mathbf{x}) := \underset{\mathbf{x} \in \mathcal{C}}{\arg\max}\ \mathbb{E}_{\theta \sim \mathcal{D}}[F_\theta(\mathbf{x})], \tag{3.8}$$

where F is submodular, monotone, DR-submodular; and $\mathcal{C} \subseteq \mathbb{R}_+^n$ is a convex set.

Since multilinear continuation preserves monotonicity and submodularity, SCG can be exploited to solve the continuous counterpart of the dependant data forgetting problem. That is, finding $\mathbf{x}^* \in \mathbb{R}_+^n$, $n = |D|$, such that

$$\mathbf{x}^* = \underset{\mathbf{x} \in \mathcal{C}}{\arg\max}\, F(\mathbf{x}),$$

where

$$\mathcal{C} = \{\mathbf{x} \in [0,1]^n\ :\ \sum_{i=1}^{n} x_i \leq B\},$$

and $F(\mathbf{x}) = \mathbb{E}_{(d,q) \sim \mathcal{D}}\, F_{q,d}(\mathbf{x})$, with $F_{q,d}(\mathbf{x})$ the multilinear extension of $f_{q,d}(D')$ as in (3.7)[2]. Both formulations lead to the same optimal value[3]. In other words,

$$\max_{\mathbf{x} \in [0,1]^n\, :\, \sum_{i=1}^n x_i \leq B} F(\mathbf{x}) = \max_{D' \subseteq D\, :\, |D'| \leq B} f(D') = OPT.$$

Hence, solving dependent data forgetting boils down to solving its continuous counterpart and properly rounding the solution.

The SCG routine (algorithm 6) yields a solution within a $(1 - 1/e - \epsilon)$ approximation factor to the optimal one, in expectation, by just using $\mathcal{O}(1/\epsilon^3)$ oracle calls to the stochastic gradients of

[2]Objective $F(\mathbf{x})$ is simply the multilinear extension of set function $f(D') = \mathbb{E}_{(d,q) \sim \mathcal{D}}\, f_{q,d}(D')$, with $f_{q,d}(\mathbf{x})$ as in (3.7). However, since the multilinear extension of any set function is itself linear, it "jumps" over the expected value operator. Consequently, $F(\mathbf{x}) = \mathbb{E}_{(d,q) \sim \mathcal{D}}\, F_{q,d}(\mathbf{x})$, with $F_{q,d}(\mathbf{x})$ the multilinear extension of $f_{q,d}(D')$ as in (3.7).

[3]This is simply due to the fact that problems 3 and 4 lead to the same optimal value for arbitrary matroids when the latter is the derived from the former via multilinear continuation [Calinescu et al.(2011)].

objective F. The algorithm builds a solution $\mathbf{x}^*$ inside the feasible set $\mathcal{C}$ starting from the origin $\mathbf{0} \in \mathbb{R}^n_+$ through a series of (gradient ascent) rounds. Each round $t = 1, \ldots, T$, consists of three simple steps: First, the estimated gradient of F at time t, denoted $\mathbf{d}_t$, is computed via the recursion

$$\mathbf{d}_t = (1 - \rho_t)\mathbf{d}_{t-1} + \rho_t \nabla F_{\theta_t}(\mathbf{x}_t),$$

where $\rho_t = \frac{4}{(t+8)^{2/3}}$, and $\mathbf{d}_0 = \mathbf{0}$. Second, the gradient ascent direction inside of $\mathcal{C}$ at time t, denoted $\mathbf{v}_t$, is defined as

$$\mathbf{v}_t = \arg\max_{\mathbf{v} \in \mathcal{C}} \{\langle \mathbf{d}_t, \mathbf{v} \rangle\};$$

(i.e., as the solution to a linear program in **v** over the convex set $\mathcal{C}$). Last, the partial solution at time t, denoted $\mathbf{x}_t$, is updated as

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \frac{1}{T}\mathbf{v}_t,$$

with $\mathbf{x}_0 = \mathbf{0}$.

Fortunately, calculating the stochastic gradients of F and solving the linear program over $\mathcal{C}$ are extremely simple tasks in the case of the continuous version of dependent data forgetting. Refer to appendix B for the complete details on such computations.

```
Algorithm 6: Stochastic Continuous Greedy (SCG)
  Data: Monotone, DR-submodular function $F(\mathbf{x}) = \mathbb{E}_{\theta \sim \mathcal{D}}(F_\theta(\mathbf{x}))$, convex set $\mathcal{C}$, and $T$.
  Result: $\mathbf{x}^* \in \mathcal{C}$ such that $\mathbb{E}[F(\mathbf{x}^*)] \geq (1 - 1/e) \cdot OPT - \mathcal{O}(\frac{1}{T^{1/3}})$, where $OPT$ is the
          value of the optimal (intractable) solution of problem 4.
  $\mathbf{d}_0 \leftarrow \mathbf{0},\ \mathbf{x}_0 \leftarrow \mathbf{0}$;
  for $t = 1, \ldots, T$ do
      $\mathbf{d}_t \leftarrow (1 - \rho_t)\mathbf{d}_{t-1} + \rho_t \nabla F_{\theta_t}(\mathbf{x}_t)$,   where $\rho_t = \frac{4}{(t+8)^{2/3}}$;
      $\mathbf{v}_t \leftarrow \arg\max_{\mathbf{v} \in \mathcal{C}} \{\langle \mathbf{d}_t, \mathbf{v} \rangle\}$;
      $\mathbf{x}_{t+1} \leftarrow \mathbf{x}_t + \frac{1}{T}\mathbf{v}_t$;
  end
  return $\mathbf{x}^* = \mathbf{x}_T$
```

3.2.3.3 The pipage rounding scheme

SCG outputs a vector $\mathbf{x}^* \in \mathcal{C}$ such that $F(\mathbf{x}^*) \geq (1 - 1/e - \epsilon) \cdot OPT$, in expectation. To map said solution back into the discrete domain **without losing the approximation factor**, we resort on the (value-preserving[4]) **randomized pipage rounding scheme** (algorithm 7)[Calinescu et al.(2011)]. Randomized pipage rounding discretizes $\mathbf{x}^*$ into a feasible corner $\mathbf{e}_{D^*}$ of the unit cube that attains a higher utility in expectation (i.e., $\mathbb{E}[F(\mathbf{e}_{D^*})] \geq F(\mathbf{x}^*)$). Due to the nature of multilinear extensions, $F(\mathbf{e}_{D^*}) = f(D^*)$ where D^* contains tuple $d_i \in D$ if the i-th component of $\mathbf{e}_{D^*}$ is equal to 1.

[4]A value-preserving rounding technique is one such that the rounded vector has at least the same utility as the continuous one.

Hence,

$$f(D^*) = F(\mathbf{e}_{D^*}) \geq F(\mathbf{x}^*) \geq (1 - 1/e - \epsilon) \cdot OPT,$$

in expectation.

Consequently, dependent data forgetting can be solved within a $(1 - 1/e - \epsilon)$ factor, in expectation, by solving its continuous counterpart through SCG and rounding the latter's output with the randomized pipage scheme.

```
Algorithm 7: Randomized pipage rounding
Data: Fractional vector $\mathbf{x} \in \mathcal{C} \subseteq [0,1]^n$.
Result: Binary vector $\mathbf{e}_{D^*} \in \mathcal{C} \subseteq [0,1]^n$ such that $\mathbb{E}[F(\mathbf{e}_{D^*})] \geq F(\mathbf{x})$.
while $\mathbf{x}$ is fractional do
    Find two fractional coordinates $x_p, x_q$;
    $\mathbf{d}_x \leftarrow \mathbf{e}_p - \mathbf{e}_q$   /* Note: $\mathbf{e}_i$ is the $i$-th canonical vector of
        $\mathbb{R}^n$, i.e., the binary vector of dimension $n$ with all $0'$s
        and a 1 in position $i$.                                   */
    $\alpha_x \leftarrow \min\{1 - x_p, x_q\}$;
    $\beta_x \leftarrow \min\{1 - x_q, x_p\}$;
    $p \leftarrow \frac{\alpha_x}{\alpha_x + \beta_x}$;
    $p' \leftarrow$ random number between 0 and 1;
    if $p' < p$ then
        $\mathbf{x} \leftarrow \mathbf{x} - \beta_x \mathbf{d}_x$   /* i.e., with probability $p$ move in the
            direction $-\beta_x \mathbf{d}_x$                         */
    end
    else
        $\mathbf{x} \leftarrow \mathbf{x} + \alpha_x \mathbf{d}_x$   /* i.e., with probability $1-p$ move in the
            direction $\alpha_x \mathbf{d}_x$                          */
    end
end
return $\mathbf{e}_{D^*} \leftarrow \mathbf{x}$;
```

3.2.4 Demonstration of the data forgetting algorithms

All in all, we have showed that **independent** data forgetting can be exactly solved in pseudo-polynomial time via the the knapsack solver with input $v_d = \mathbb{E}_{q \sim \mathcal{Q}}\left[\frac{|q(\{d\})|}{|q(D)|}\right]$, $w_d = 1$, $W = B$. Furthermore, we have proved that **dependent** data forgetting can be solved within a $(1 - 1/e - \epsilon)$ factor, in expectation, by lifting the problem into the continuous domain via multilinear continuation, solving its continuous counterpart through SCG, and rounding the latter's output with the randomized pipage scheme. In the following, we will refer to the independent (resp. dependent) data forgetting routine as `IndepDF` (resp. `DepDF`). Finally, we exhibit the execution of both routines on toy example 1.0.1 to provide the reader an even crisper understanding.

3.2.4.1 Setting:

Input

1. Database $D = \{d_1, d_2, d_3\}$ where

id	Name	Last name	Age	City	Job	Married	Children
d_1	James	Smith	25	Amsterdam	Teacher	Yes	2
d_2	Wade	Brown	25	Paris	Firefighter	No	0
d_3	Olivia	Smith	25	Amsterdam	Teacher	Yes	2

2. Query-log $Q = (q_1, q_2, q_1, q_3, q_3, q_3, q_2, q_1, q_2)$, where

   · q_1 : Age = 25,

   · q_2 : City = Amsterdam $\wedge$ Job = Teacher,

   · q_3 : Name = Olivia.

   The order of appearance of the queries in Q indicate the order in which they have been posed. In this case, the first query in the query-log is q_1 and the last one q_2.

3. Budget $B = 2$.

Input pre-processing Query-log Q is pre-processed to extract the set of unique queries, also denoted Q (notation abuse), together with their relative frequencies (i.e., empirical distribution $\mathcal{Q}$). In this simple case, the set of unique queries contains only three elements (namely, q_1, q_2, q_3), and $\mathcal{Q}$ is the uniform distribution $\mathcal{U}$ with domain $\{q_1, q_2, q_3\}$. That is,

$$(Q, \mathcal{Q}) = (\{q_1, q_2, q_3\}, \quad \mathcal{U}(q_1, q_2, q_3)).$$

The relevant answer sets are $q_1(D) = \{d_1, d_2, d_3\},\ q_2(D) = \{d_1, d_3\},\ q_3(D) = \{d_3\}$.

Optimization objective We aim finding a subset $D^* \subseteq D$ such that

$$D^* = \underset{D' \subseteq D,\ |D'| \leq 2}{\arg\max} \quad f(D'), \tag{3.9}$$

where

$$f(D') = \mathbb{E}_{(d,q)\sim\mathcal{D}}\, f_{q,d}(D') = \mathbb{E}_{(d,q)\sim\mathcal{D}} \max_{d' \in q(D')} \mathrm{sim}(d, d')$$

with $\mathrm{sim}(d, d') = Jaccard(d, d')$ as in (3.4).

3.2.4.2 `DepDF`

Multilinear continuation The multilinear extension of $f(D')$ is the continuous function $F(\mathbf{x}) = \mathbb{E}_{(d,q)\sim\mathcal{D}}\, F_{q,d}(\mathbf{x})$, where $F_{q,d}(\mathbf{x})$ is the multilinear extension of $f_{q,d}(D') = \max_{d' \in q(D')} \mathrm{sim}(d, d')$.

SCG The SCG algorithm is ran 100 times for $T = 100$ iterations on the multilinear extension $F(\mathbf{x})$ and convex set $\mathcal{C} = \{\mathbf{x} \in [0,1]^3 \ : \ \sum_{i=1}^{3} x_i \leq 2\}$. Figure 3.2 exhibits the partial solutions $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_T = \mathbf{x}^*$ generated by each one of the SCG executions.

The power set of $D = \{d_1, d_2, d_3\}$ is identified with the corners of the unit cube in $\mathbb{R}^3$ as follows: $(0,0,0) = \emptyset$, $(1,0,0) = \{d_1\}$, $(0,1,0) = \{d_2\}$, $(0,0,1) = \{d_3\}$, $(1,1,0) = \{d_1, d_2\}$, $(0,1,1) = \{d_2, d_3\}$, $(1,0,1) = \{d_1, d_3\}$, and $(1,1,1) = D$. Starting from the origin $\mathbf{x}_0$, SCG navigates through the continuous space by making very small steps in the estimated direction of the gradient $\nabla F(\mathbf{x})$. Namely, steps of size $1/T = 1/100$ in the direction parallel to the estimated gradient $\mathbf{d}_t$ inside of $\mathcal{C}$, i.e., vector $\mathbf{v}_t$ (see algorithm 6) [5].

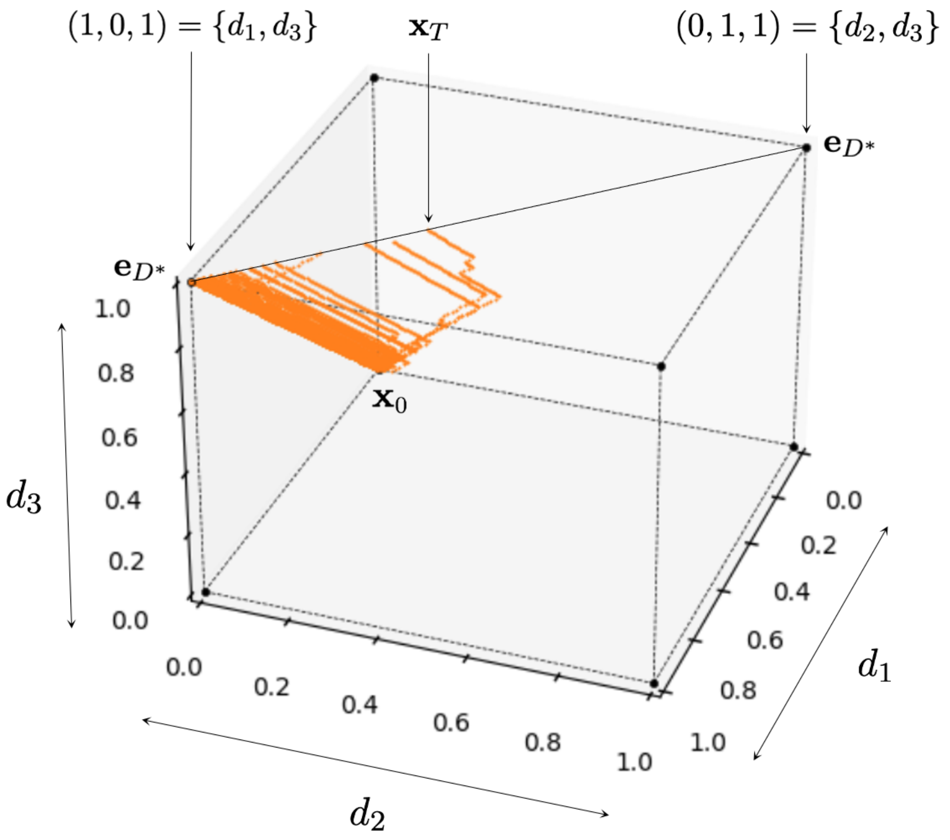


Figure 3.2: Depiction of 100 partial solutions $\mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_{100}$ (in orange) generated by each one of the 100 executions of the SCG algorithm given $F(\mathbf{x}) = \mathbb{E}_{(d,q)\sim\mathcal{D}} F_{q,d}(\mathbf{x})$, where $F_{q,d}(\mathbf{x})$ is the multilinear extension of $f_{q,d}(D')$ as in (3.7), and $\mathcal{C} = \{\mathbf{x} \in [0,1]^3 \ : \ \sum_{i=1}^{3} x_i \leq 2\}$. Vector $\mathbf{e}_{D^*}$ denotes the outcome of the Pipage rounding scheme. Vector $\mathbf{e}_{D^*} = (0,1,1)$ for executions $3, 11, 89, 91$, and $\mathbf{e}_{D^*} = (1,0,1)$ for the remaining runs.

[5] The fact that each step is taken in the direction of $\mathbf{v}_t$ is a vital requirement to guarantee that $\mathbf{x}_T = \mathbf{x}_{100} = \mathbf{x}^*$ will be inside of the feasible set $\mathcal{C}$.

Pipage rounding Each one of the SCG runs returns a solution $\mathbf{x}^* = \mathbf{x}_T$ in the border of $\mathcal{C}$, represented by the diagonal line on the top face of the cube in figure 3.2. Pipage rounding discretizes $\mathbf{x}^*$ into a corner of the unit cube without losing utility in expectation. Out of the 100 runs, vector $\mathbf{e}_{D^*} = (0, 1, 1)$ for executions $3, 11, 89, 91$, and $\mathbf{e}_{D^*} = (1, 0, 1)$ for the remaining runs.

Output Solution D^* can be read from the non-zero entries of $\mathbf{e}_{D^*}$. In the case of executions $3, 11, 89, 91$, as only the second and third components of $\mathbf{e}_{D^*}$ are non-zero, the retained subset of tuples would be

$$D^* = \{d_2, d_3\} \subseteq D.$$

For the contrary, in the rest of the executions, as only the first and third components of $\mathbf{e}_{D^*}$ are non-zero, the retained subset of tuples would be

$$D^* = \{d_1, d_3\} \subseteq D.$$

According to objective (3.9),

$$f(\{d_2, d_3\}) = 0.9306,$$

and

$$f(\{d_1, d_3\}) = 0.8974.$$

3.2.4.3 `IndepDF`

Value computations Each tuple $d \in D$ is assigned value $v_d = \mathbb{E}_{q \sim \mathcal{Q}} \left[\frac{|q(\{d\})|}{|q(D)|} \right]$. That is,

$$\begin{aligned} v_{d_1} &= \frac{1}{3} \cdot \frac{1}{3} + \frac{1}{3} \cdot \frac{1}{2} = 0.28, \\ v_{d_2} &= \frac{1}{3} \cdot \frac{1}{3} = 0.11, \\ v_{d_3} &= \frac{1}{3} \cdot \frac{1}{3} + \frac{1}{3} \cdot \frac{1}{2} + \frac{1}{3} = 0.61. \end{aligned}$$

Knapsack solver The Kanapsack solver is feed input $v_{d_1} = 0.28, v_{d_2} = 0.11, v_{d_3} = 0.61$ (values); $w_{d_1} = w_{d_2} = w_{d_3} = 1$ (weights); and $W = 2$ (capacity). Since all weights are equal, the two most valuable points get returned.

Output The retained subset of tuples correspond to the two most frequently retrieved datapoints (recall example 1.0.1). That is,

$$D^* = \{d_1, d_3\} \subseteq D,$$

which, according to objective (3.9), has a utility of 0.8974.

Chapter 4

Experimental evaluation

4.1 Experimental setup

We tested `IndepDF` and `DepDF` on real and synthetic datasets against a baseline algorithm using a server with 250 GB RAM and 112 cores.

4.1.1 Baseline algorithm

- `LAZY GREEDY`: Our problem's objective (in both flavours) is the expected value of a non-negative monotone submodular function, and hence, it's non-negative monotone submodular itself. Consequently, the classic `GREEDY` algorithm (algorithm 1)[Nemhauser et al.(1978)] or its popular accelerated version `LAZY GREEDY` (algorithm 2) [Minoux(1978)] guarantees a solution with a $(1 - 1/e)$ approximation factor.

4.1.2 Datasets

4.1.2.1 Synthetic data

With respect to the data, we created the nested sequence of datasets $D_{1K} \subseteq D_{10K} \subseteq D_{50K} \subseteq D_{100K}$. Dataset D_{1K} is made up of 1.000 tuples of size 101 organized into 10 groups. Tuples are approximately 100% dissimilar between groups and approximately 100% similar within groups. More precisely, group $i \in \{0, \ldots, 9\}$ contains 100 tuples of the form t^i_α where

$$t^i_\alpha = (i, \ldots, i, \alpha) \in \mathbb{R}^{101},\ \alpha = \text{random}([0, 1]). \tag{4.1}$$

Furthermore, datasets $D_{10K}, D_{50K}, D_{100K}$ are simple extensions of D_{1K}. Specifically, D_{10K} (resp. D_{50K}, D_{100K}) shares the first 1.000 tuples with D_{1K}. The remaining 9.000 (resp. $49.000, 99.000$) tuples correspond to 9.000 (resp $49.000, 99.000$) tuples of the form t^{10}_α as in (4.1).

Regarding the queries, we generated the nested sequence of query-logs $Q_1 \subseteq Q_{100} \subseteq Q_{1K} \subseteq Q_{10K}$. Log Q_1 contains a single query q with probability one retrieving every tuple in D_{1K}. Moreover, Q_{100}, Q_{1K}, Q_{10K} extend Q_1. Specifically, Q_{100} (resp. Q_{1K}, Q_{10K}) shares the first query with

Q_1. The remaining 99 (resp 999, 9.999) queries retrieve 100 random points of D_{1K} with probability $\epsilon \approx 0$. That is, they are of the form

$$q(D_{1K}) = \{\alpha_1, \dots, \alpha_{100}\} \subseteq D_{1K},\ \alpha_i = \text{random\_int}([1, 1.000]).$$

4.1.2.2 Real data

We exploited real datasets coming from three different sources with corresponding real query-logs.

· **Flights**: As a small-size real database instance, we considered the `flights.csv` file inside the `flight_2` folder of the spider dataset [1]. Its query-log corresponds to all of the queries that refer to the `flights.csv` table in files `dev_gold.sql` and `train_gold.sql`, also from the spider dataset.

· **SkyServer**: As a medium-size real database instance, we selected the subset of the `PhotoTag` table in sky server retrieved by the queries made by users between $01/01/2023 - 31/01/2023$ via the sky server SQL query service. Naturally, the query-log corresponds to the portion of queries executed between $01/01/2023 - 31/01/2023$ that refer to the `PhotoTag` table. The latter con be downloaded from the sky server SQL log[2].

· **Wikidata**: As a large-size real database instance, we choose the portion of wikidata retrieved by the SPARQL queries [3] executed by users via the wikidata query service between $12/06/2017 - 09/07/2017$ [4]. Due to the nature of the data, some pre-processing was required: Every webpage is identified by a unique id, and any two webpages only share the "label" and "description" fields. Hence, given any id, we concatenated its corresponding webpage label and description texts into a single sentence and embedded the latter using the "all-MiniLM-L6-v2" sentence transformer. This way, every webpage is identified with a vector in $\mathbb{R}^{384}$.

Dataset name	Number of rows	Number of columns	Number of queries	Nature
D_{flights}	1.200	4	37	categorical
$D_{\text{skyserver}}$	11.058	97	11.058	numeric
D_{wikidata}	131.148	384	14.723	numeric

Table 4.1: Characteristics of the considered real datasets.

[1]https:drive.google.comuc?export=download&id=1TqleXec_OykOYFREKKtschzY29dUcVAQ

[2]https://skyserver.sdss.org/log/en/traffic/sql.asp

[3]We only kept those queries whose answer sets were smaller than 100

[4]https:analytics.wikimedia.orgdatasetsone-offwikidatasparql_query_logs2017-06-12_2017-07-09/2017-06-12_2017-07-09_organic.tsv.gz

4.2 Experimental results

4.2.1 Quality

The objective of the first group of experiments threefold: First, checking whether `DepDF` is able to fetch a solution with maximal utility in a scenario where such solution exits; Second, studying how the quality of the produced solutions by `IndepDF`,`DepDF`, and `LAZY GREEDY` increase as the budget size increases; Third, comparing the quality of the produced solutions by `DepDF` for different configurations of its hyperparameter T, and that of `IndepDF` and `LAZY GREEDY`.

4.2.1.1 Recovering ground truth

Recall that dataset D_{1K} is made up of 1.000 tuples organized into 10 groups where tuples are approximately 100% dissimilar between groups and approximately 100% similar within groups. Furthermore, Q_1 contains a single query that retrieves the full dataset (i.e., $q(D_{1K}) = D_{1K}$). In this fabricated setting, any subset of D_{1K} that includes at least one tuple from each one of the 10 groups is an optimal solution to the dependent data forgetting problem for sim$(\cdot,\cdot) = Jaccard(\cdot,\cdot)$ as in 3.4. Hence, for any budget $B \geq 10$, there exists at least one optimal solution. Table 4.2 shows the utility of the solutions constructed by `IndepDF`, `DepDF`, and `LAZY GREEDY` in this artificial setting for budget $B = 20 = 2\% \cdot |D_{1K}|$.

	`IndepDF`	`DepDF`	`LAZY GREEDY`
Best utility	0.1965	**0.9808**	**0.9808**
Average utility	0.1965	0.9709	**0.9808**
Standard deviation	0	0.0295	0

Table 4.2: Utility of `IndepDF`, `DepDF` (with $T = 2.000$), and `LAZY GREEDY` for dataset D_{1K}, query-log Q_1, and budget $B = 20$. As the theoretical guarantees of `DepDF` are given in expectation, the reported results are provided over 100 runs of the algorithm.

4.2.1.2 Solution quality when increasing budget size

Four scenarios were considered:

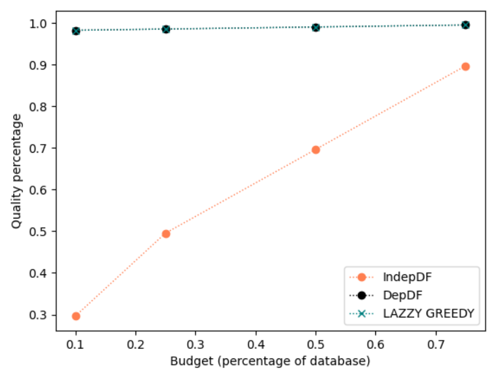
(a) Dataset D_{1K}, query-log Q_1, and $B = 10\%, 25\%, 50\%, 75\%$ of $|D_{1K}| = 1.000$.

(b) Real dataset D_{flights}, query-log Q_{flights}, and $B = 10\%, 25\%, 50\%, 75\%$ of $|D_{\text{flights}}| = 1.200$.

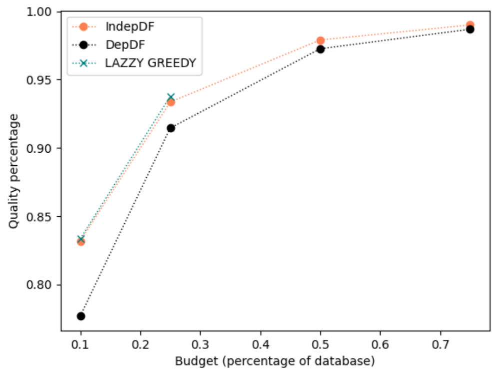
(c) Dataset $D_{\text{skyserver}}$,query-log $Q_{\text{skyserver}}$, and $B = 10\%, 25\%, 50\%, 75\%$ of $|D_{\text{skyserver}}| = 11.058$.

(d) Dataset D_{wikidata}, query-log Q_{wikidata}, and $B = 1\%, 10\%, 25\%, 50\%$ of $|D_{\text{wikidata}}| = 131.148$.

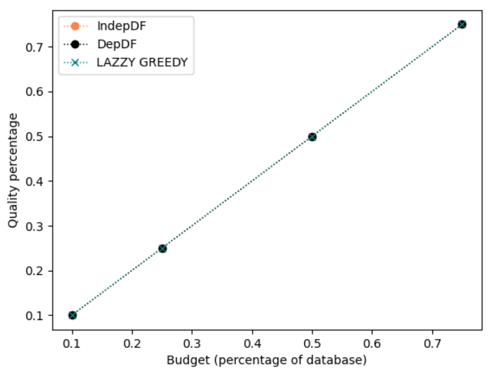
The results are depicted in figure 4.1



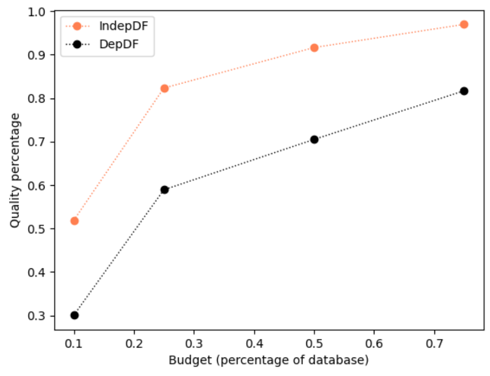
(a) Synthetic data.



(b) Flights.



(c) Skyserver.



(d) Wikidata.

Figure 4.1: Utility of `IndepDF`, `DepDF` (with $T = 2.000$), and `LAZY GREEDY` when changing budget size. Due to the nature of the data, $\text{sim}(\cdot,\cdot) = Jaccard(\cdot,\cdot)$ as in 3.4 for scenarios (a) and (b), and $\text{sim}(\cdot,\cdot) = cos(\cdot,\cdot)$ as in 3.5 for scenarios (c) and (d). As the theoretical guarantees of `DepDF` are given in expectation, the best quality solution over 100 runs of the algorithm is reported. All graphs are plotted on logarithmic scale.

4.2.1.3 Solution quality when increasing the number of iterations T

Four scenarios were considered:

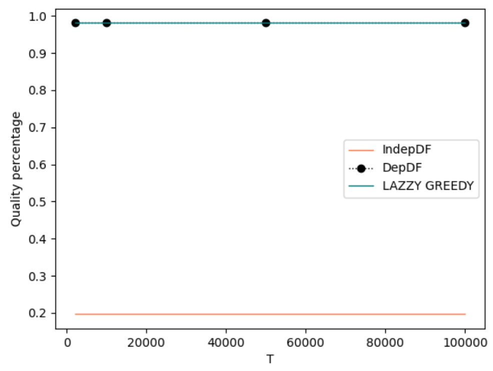
(a) Dataset D_{1K}, query-log Q_1, $B = 20$, and $T = 2K, 10K, 50K, 100K$.

(b) Dataset D_{flights}, query-log Q_{flights}, $B = 300 = 25\% \cdot |D_{\text{flights}}|$, and $T = 2K, 10K, 50K, 100K$.

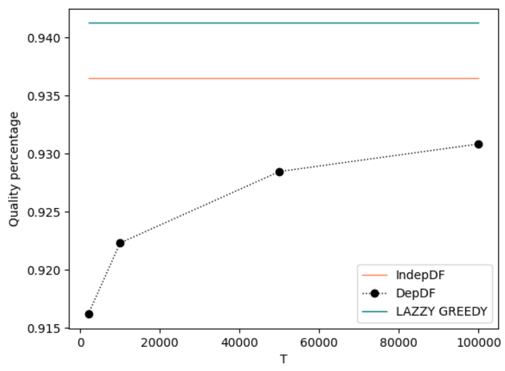
(c) Dataset $D_{\text{skyserver}}$, query-log $Q_{\text{skyserver}}$, $B = 2764 = 25\% \cdot |D_{\text{skyserver}}|$, and $T = 2K, 10K, 50K, 100K$.

(d) Dataset D_{wikidata}, query-log Q_{wikidata}, $B = 1311 = 1\% \cdot |D_{\text{wikidata}}|$, and $T = 2K, 10K, 50K, 100K$.

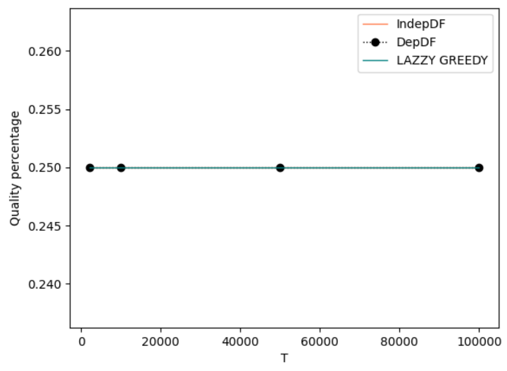
Figure 4.2 depicts the results.



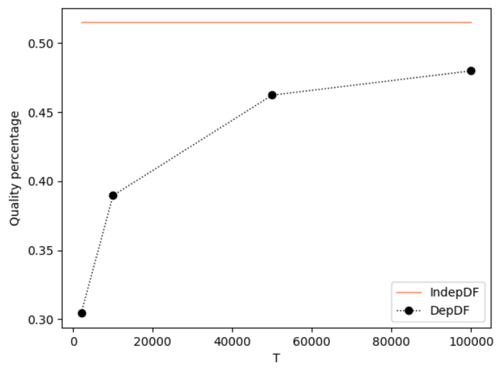
(a) Synthetic data.



(b) Flights.



(c) Skyserver.



(d) Wikidata.

Figure 4.2: Utility of `IndepDF`, `DepDF` (with increasing values of hyperparameter T), and `LAZY GREEDY`. Due to the nature of the data, $\text{sim}(\cdot,\cdot) = Jaccard(\cdot,\cdot)$ as in 3.4 for scenarios (a) and (b), and $\text{sim}(\cdot,\cdot) = cos(\cdot,\cdot)$ as in 3.5 for scenarios (c) and (d). As the theoretical guarantees of `DepDF` are given in expectation, the best quality solution over 100 runs of the algorithm is reported. All graphs are plotted on logarithmic scale.

4.2.2 Time performance

The objective of the second group of experiments is studying how the time performance of our proposed data forgetting routines compares to that of `LAZY GREEDY` as the dataset, query-log, and budget size increases.

4.2.2.1 Time performance when increasing dataset size

Four scenarios were considered:

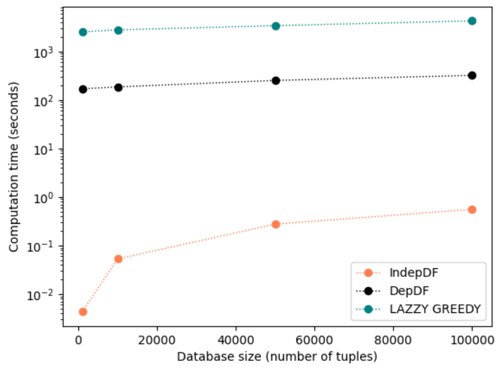
(a) Datasets $D_{1K} \subseteq D_{10K} \subseteq D_{50K} \subseteq D_{100K}$, query-log Q_1, and budget $B = 20$.

(b) Datasets $25\%, 50\%, 75\%, 100\%$ of D_{flights}, query-log Q_{flights}, and budget $B = 120$.

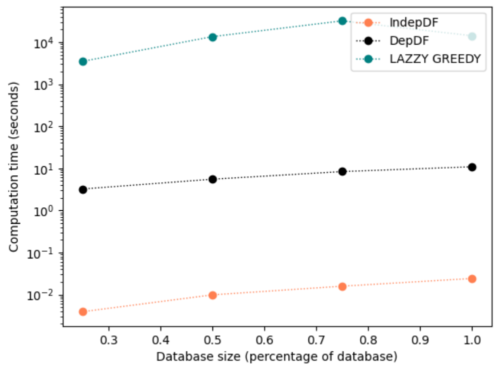
(c) Datasets $25\%, 50\%, 75\%, 100\%$ of $D_{\text{skyserver}}$, query-log $Q_{\text{skyserver}}$, and budget $B = 1105$.

(d) Datasets $25\%, 50\%, 75\%, 100\%$ of D_{wikidata}, query-log Q_{wikidata}, and budget $B = 1311$.

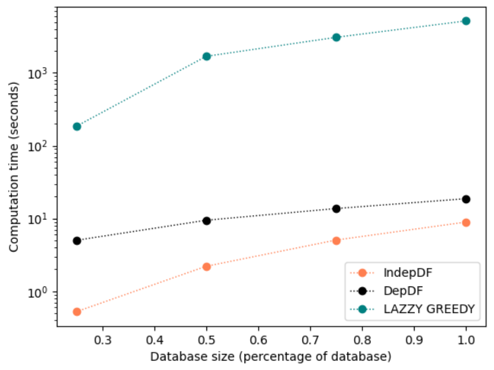
The results are depicted in figure 4.3.



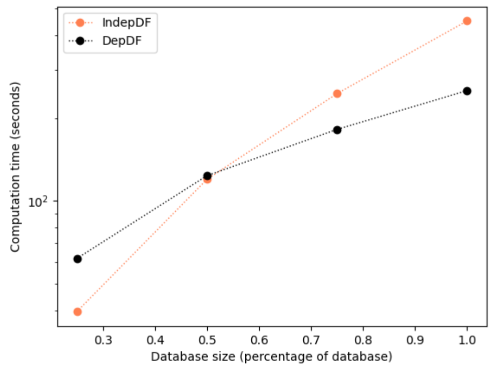
(a) Synthetic data.



(b) Flights.



(c) Skyserver.



(d) Wikidata.

Figure 4.3: Time performance of `IndepDF`, `DepDF` (with $T = 2.000$), and `LAZY GREEDY` when increasing the dataset size. Due to the nature of the data, $\text{sim}(\cdot,\cdot) = Jaccard(\cdot,\cdot)$ as in 3.4 for scenarios (a) and (b), and $\text{sim}(\cdot,\cdot) = cos(\cdot,\cdot)$ as in 3.5 for scenarios (c) and (d). As the theoretical guarantees of `DepDF` are given in expectation, the average time performance over 100 runs of the algorithm is reported. All graphs are plotted on logarithmic scale.

4.2.2.2 Time performance when increasing query-log size.

Four scenarios were considered:

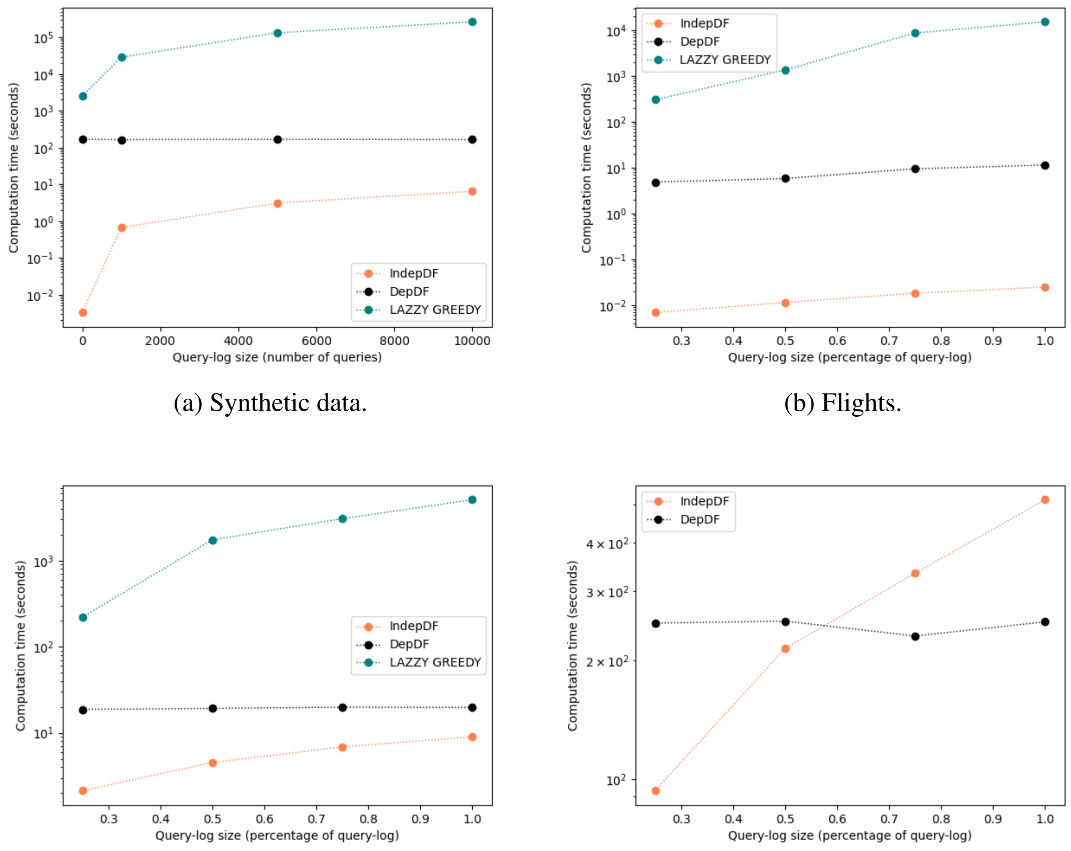
(a) Dataset D_{1K}, query-logs $Q_1 \subseteq Q_{100} \subseteq Q_{1K} \subseteq Q_{10K}$, and budget $B = 20$.

(b) Dataset D_{flights}, query-logs $25\%, 50\%, 75\%, 100\%$ of Q_{flights}, and budget $B = 120$.

(c) Dataset $D_{\text{skyserver}}$, query-logs $25\%, 50\%, 75\%, 100\%$ of $Q_{\text{skyserver}}$, and budget $B = 1105$.

(d) Dataset D_{wikidata}, query-logs $25\%, 50\%, 75\%, 100\%$ of Q_{wikidata}, and budget $B = 1311$.

The results are plotted in figure 4.4.



(a) Synthetic data. (b) Flights.

(c) Skyserver. (d) Wikidata.

Figure 4.4: Time performance of `IndepDF`, `DepDF` (with $T = 2.000$), and `LAZY GREEDY` when increasing the query-log size. Due to the nature of the data, $\text{sim}(\cdot,\cdot) = Jaccard(\cdot,\cdot)$ as in 3.4 for scenarios (a) and (b), and $\text{sim}(\cdot,\cdot) = cos(\cdot,\cdot)$ as in 3.5 for scenarios (c) and (d). As the theoretical guarantees of `DepDF` are given in expectation, the average time performance over 100 runs of the algorithm is reported. All graphs are plotted on logarithmic scale.

4.2.2.3 Time performance when increasing budget size

Four scenarios were considered:

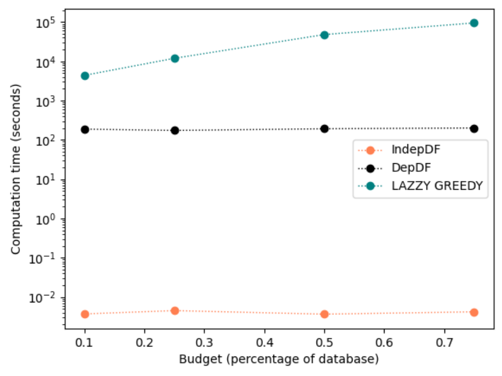
(a) Dataset D_{1K}, query-log Q_1, and $B = 10\%, 25\%, 50\%, 75\%$ of $|D_{1K}| = 1.000$.

(b) Dataset D_{flights}, query-log Q_{flights}, and $B = 10\%, 25\%, 50\%, 75\%$ of $|D_{\text{flights}}| = 1.200$.

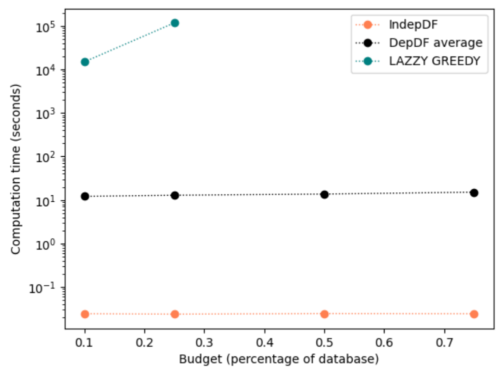
(c) Dataset $D_{\text{skyserver}}$,query-log $Q_{\text{skyserver}}$, and $B = 10\%, 25\%, 50\%, 75\%$ of $|D_{\text{skyserver}}| = 11.058$.

(d) Dataset D_{wikidata}, query-log Q_{wikidata}, and $B = 1\%, 10\%, 25\%, 50\%$ of $|D_{\text{skyserver}}| = 131.148$.

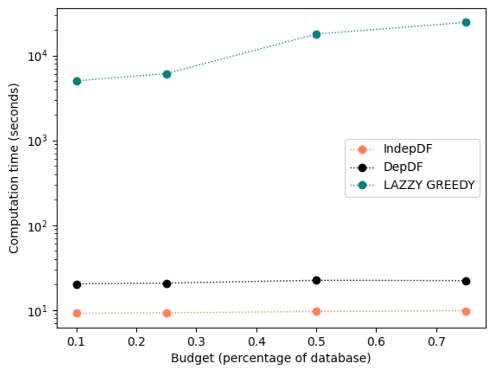
The results can be found in figure 4.5.



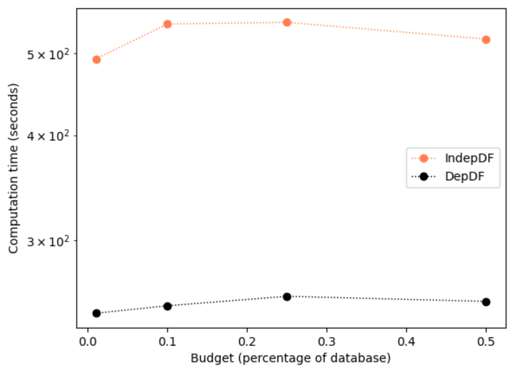
(a) Synthetic data.



(b) Flights.



(c) Skyserver.



(d) Wikidata.

Figure 4.5: Time performance of `IndepDF`, `DepDF` (with $T = 2.000$), and `LAZY GREEDY` when increasing the budget size. Due to the nature of the data, sim$(\cdot,\cdot) = Jaccard(\cdot,\cdot)$ as in 3.4 for scenarios (a) and (b), and sim$(\cdot,\cdot) = cos(\cdot,\cdot)$ as in 3.5 for scenarios (c) and (d). As the theoretical guarantees of `DepDF` are given in expectation, the average time performance over 100 runs of the algorithm is reported. All graphs are plotted on logarithmic scale.

4.3 Discussion

4.3.1 Quality

First, we observe that answer set diversity plays a significant role in the quality performance of our proposed data forgetting routines. As shown in table 4.3, answer sets are heterogeneous in the smallest synthetic dataset and homogeneous in all three real datasets. We note that `DepDF` performs equal to `LAZY GREEDY` and significantly better than `IndepDF` when answer sets are very diverse (see table 4.2 and figure 4.1 (a)). For the contrary, if answer sets are not diverse, `IndepDF` performs equal to `LAZY GREEDY`[5] and slightly better than `DepDF` (see figure 4.1 (b),(d)). Despite surprising, this behaviour was expected. If answer sets are heterogeneous, seeking a solution that

[5]Experiments were left running for 3 days. `LAZY GREEDY` did not finish in some of the experiments.

diversely covers them, as `DepDF` or `LAZY GREEDY`, should attain the highest utility. For the contrary, if answer sets are homogeneous, each datapoint within the answer set is (essentially) of the same importance to the latter. Hence, in this scenario, the optimization objective converges to that of independent data forgetting; and, consequently we expect `IndepDF` to perform the best.

Data	(D_{1K}, Q_1)	$(D_{\text{flights}}, Q_{\text{flights}})$	$(D_{\text{skyserver}}, Q_{\text{skyserver}})$	$(D_{\text{wikidata}}, Q_{\text{wikidata}})$
$av\_std(q(D))$	0.2961	0.0744	0	0.0965

Table 4.3: Average standard deviation of pairwise similarities between tuples inside answer sets. Given a dataset D and a query-log Q, answer set diversity can be measured as follows: Compute $q(D)$ for all $q \in Q$; next, compute all pairwise similarities between tuples $d \in q(D)$ collecting them in a list; subsequently, compute the standard deviation of each one of the $|Q|$ lists; last, average over all computed standard deviations. This final quantity, denoted $av\_std(q(D))$, indicates the average diversity over all answer sets $q(D)$ for a given dataset D and query-log Q.

Second, we notice that budget size B has a major impact on the quality performance of the three algorithms. We observe that the quality of the produced solution approaches 1 as $B \to |D|$ for all routines across all datasets (see figure 4.1). Again, this is expected behaviour, and simply a consequence of the monotonicity of our optimization objective (i.e, the fact that $f(D'') \leq f(D')$ if $D'' \subseteq D'$).

Third, we note that hyperparameter T significantly affects the quality performance of `DepDF`. Specifically, the quality of the produced solution grows as $T \to \infty$ in a diminishing returns fashion (see figure 4.2). Said conduct agrees with the fact that `DepDF` produces an approximate solution with an error term that decreases in $\mathcal{O}(1/T^{1/3})$, where T is the number of SCG iterations inside the `DepDF` routine.

Last, it's worth pointing out that input $(D_{\text{skyserver}}, Q_{\text{skyserver}})$ is a bit peculiar. Each tuple in $D_{\text{skyserver}}$ is retrieved exactly once by a query of the form "q : id $= n$" for $1 \leq n \leq |D_{\text{skyserver}}|$. Consequently, $|q(D)| = 1$ for all $q \in Q_{\text{skyserver}}$ and every subset of $D_{\text{skyserver}}$ with cardinality B attains the same (optimal) utility (see the overlapping graphs in figures 4.1 (c), and 4.2 (c)).

4.3.2 Time performance

First, we observe that `IndepDF` and `DepDF` enjoy an extremely superior time performance to that of `LAZY GREEDY` across all datasets (see figure 4.3). Nonetheless, the time performance of all three algorithms decreases as the size of the database increases. This result should come as no surprise. In fact, `IndepDF` performs a value-assigning operation for each $d \in D$. Hence, the larger the database the more operations such algorithm conducts. Moreover, `DepDF` performs T iterations each requiring the computation of $\nabla F(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^{|D|}$ and F is the multilinear extension of f. Therefore, the larger the database, the more partial derivatives must be computed per iteration. Furthermore, `LAZY GREEDY` performs B rounds, each requiring at most $|D|$ oracle calls to objective f. Consequently, the bigger the database, the longer the routine will take per iteration.

Second, we notice that query-log size has a major impact on the time performance of `LAZY GREEDY`, a minor impact on that of `IndepDF`, and no impact on that of `DepDF` (refer to figure 4.4). Again, a very natural result. `LAZY GREEDY` requires oracle access to objective $f(D') = \mathbb{E}_{q\sim\mathcal{Q}}\mathbb{E}_{d\sim\mathcal{U}(D)} \max_{d'\in q(D')} \text{sim}(d,d')$ several times per round. Due to the $\mathcal{Q}$-dependence, such function evaluations become increasingly expensive as $|Q|$ grows. Similarly, `IndepDF` carries out computations that depend on the size of Q. Namely, each $d \in D$ gets assigned a value $v_d = \mathbb{E}_{q\sim\mathcal{Q}}\left[\frac{|q(\{d\})|}{|q(D)|}\right]$. Nonetheless, the evaluation of this value-assigning operations is orders of magnitude faster than that of objective f, explaining the time difference between the two routines. For the contrary, `DepDF`'s computation time is completely unaffected by changes in $|Q|$. The algorithm performs T rounds, each requiring the computation of $\nabla F(\mathbf{x})$, where $\mathbf{x} \in \mathbb{R}^{|D|}$ and F is the multilinear extension of f. However, as detailed in appendix B, (approximately) calculating $\nabla F(\mathbf{x})$ boils down to sampling **one query** $q \sim \mathcal{Q}$, one tuple $d \sim \mathcal{U}(q(D))$, and computing $\nabla F_{q,d}(\mathbf{x})$, where $F_{q,d}(\mathbf{x})$ is the multilinear extension of $f_{q,d}(D') = \max_{d'\in q(D')} \text{sim}(d,d')$. Hence, `DepDF` is time performance-wise-oblivious to changes in Q.

Last, we note that budget size B heavily affects the time performance of `LAZY GREEDY` and has no effect at all on our proposed data forgetting routines (see figure 4.5). Indeed, an expected outcome. As aforementioned, `LAZY GREEDY` requires B iterations. Therefore, an increase in the budget size results in an increase on the number of iterations; and by default, in the overall computation time. Conversely, `IndepDF` and `DepDF` perform the exact same number of computations independent of B. In particular, `IndepDF` performs $|D|$ operations involving Q exclusively, and `DepDF` conducts T rounds that only require access to a single query and its answer set.

4.4 Concluding remarks

In this paper we address the data reduction challenge that many organizations are inevitably facing. We formalize the problem (in the context of relational data) as a stochastic subset selection exercise, and study its two most natural variants: independent, where the topology of D is ignored, and dependent, where the structure of the data is taken into account. Our contribution revolves around two key observations. First, independent data forgetting is an instance of the 0-1 Knapsack problem; and second, dependent data forgetting can be cast as an instance of the discrete stochastic submodular maximization problem under the facility location function. Capitalizing on this connections, two novel data forgetting routines with strong theoretical guarantees are proposed: `IndepDF` and `DepDF`.

The experimental results confirm our proposed data forgetting routines' theoretical guarantees. `IndepDF` (resp. `DepDF`) has proven to produce the best quality solutions in contexts of low (resp. high) answer-set diversity. But, most importantly, both routines generate solutions as good as those yield by `LAZY GREEDY` quality-wise, but significantly faster in terms of time. In fact, `LAZY GREEDY` failed to complete any of the experiments involving Wikidata over the course of three days, whereas `IndepDF` and `DepDF` only took a few minutes. In addition, our algorithms enjoy some very desirable properties that allow them to scale up quite well. Specifically, `IndepDF`'s (resp. `DepDF`'s) time performance is only mildly affected by increases in the database size, mildly affected (resp. completely unaffected) by increases in the query-log size, and completely unaffected by budget size increases. To the best of our knowledge, `IndepDF` and `DepDF` are the first ever procedures that allow forgetting relational data in data-intensive environments, based on solid theoretical foundations.

Appendix A

Supplementary material of section 3.2.2.

A.1 Properties of function (3.3)

Proposition A.1.1. *Given a dataset D and subsets $A, B \subseteq D$,*

$$S(A,B) = \frac{1}{|A|} \cdot \sum_{d \in A} \max_{d' \in B} sim(d,d')$$

is normalized, non-negative, self-similar, but non-symmetric. That is, it satisfies all the axioms from definition 3.1.1 except symmetry. Hence, it can be regarded as a weak set similarity function.

Proof.

· **Normalized non-negativity:** The non-negativity follows from the fact that $\text{sim}(\cdot,\cdot)$ is non negative. Furthermore, since $\text{sim}(\cdot,\cdot) \in [0,1]$, each term in the sum is bounded by 1. Since there are $|A|$ terms, the total sum is bounded by $|A|$. Dividing by this very quantity, the complete expression is bounded by 1. Hence, $0 \leq S(A,B) \leq 1$.

· **Self similarity:**

$$\begin{aligned} S(A,A) &= \frac{1}{|A|} \cdot \sum_{d \in A} \max_{d' \in A} \text{sim}(d,d') \\ &= \frac{1}{|A|} \cdot \sum_{d \in A} \text{sim}(d,d) \\ &= \frac{1}{|A|} \cdot \sum_{d \in A} 1 = \frac{|A|}{|A|} = 1. \end{aligned}$$

· **Non-symmetry:** It suffices to consider input subsets such that $B \subseteq A$. In this case $S(A,B) \leq 1$, while $S(B,A) = 1$. Hence, symmetry is not guaranteed.

□

A.2 Non-negativity, monotonicity and submodularity of $f_{q,d}(D')$

Proposition A.2.1. *For any query q and datasets $D' \subseteq D$, set functions*

$$f_{q,d}(D') = \max_{d' \in q(D')} sim(d, d'),$$

are non-negative, monotone, and submodular.

Proof.

· **Normalized non-negativity:** This is an immediate consequence of the fact that

$$\text{sim}(d, d') \in [0, 1] \; \forall \, d, d'.$$

· **Monotonicity:** If $A \subseteq B \subseteq D \implies q(A) \subseteq q(B) \subseteq q(D) \implies$

$$f_q(A) = \max_{d' \in q(A)} \text{sim}(d, d') \underset{q(A) \subseteq q(B)}{\leq} \max_{d' \in q(B)} \text{sim}(d, d') = f_q(B).$$

· **Submodularity:** We rely on the definition in terms of the discrete derivative. Given any $A \subseteq B \subseteq D$ and $d^* \in D \setminus B$, we distinguish two cases:

  · **Case 1:** $q(\{d^*\}) = \emptyset$. In this scenario,

$$\begin{aligned} \Delta_{f_q}(d^*|B) &= \max_{d' \in q(B \cup \{d^*\})} \text{sim}(d, d') - \max_{d' \in q(B)} \text{sim}(d, d') \\ &\underset{q(B \cup \{d^*\}) = q(B) \cup q(\{d^*\}) = q(B)}{=} 0 \\ &\underset{q(A \cup \{d^*\}) = q(A) \cup q(\{d^*\}) = q(A)}{=} \Delta_{f_q}(d^*|A). \end{aligned}$$

  · **Case 2:** $q(\{d^*\}) = d^*$. In this scenario,

$$\begin{aligned} \Delta_{f_q}(d^*|B) &= \max_{d' \in q(B \cup \{d^*\})} \text{sim}(d, d') - \max_{d' \in q(B)} \text{sim}(d, d') \\ &= \max\Big\{0, \text{sim}(d, d^*) - \max_{d' \in q(B)} \text{sim}(d, d^*)\Big\}. \end{aligned}$$

  Now, for any $d \in D$,

$$\begin{aligned} A \subseteq B \implies q(A) \subseteq q(B) &\implies \max_{d' \in q(A)} \text{sim}(d, d') \leq \max_{d' \in q(B)} \text{sim}(d, d') \\ &\implies -\max_{d' \in q(B)} \text{sim}(d, d') \leq -\max_{d' \in q(A)} \text{sim}(d, d') \\ &\implies \text{sim}(d, d^*) - \max_{d' \in q(B)} \text{sim}(d, d') \leq \text{sim}(d, d^*) - \max_{d' \in q(A)} \text{sim}(d, d'). \end{aligned}$$

  Hence, $\Delta_{f_q}(d^*|B) \leq \Delta_{f_q}(d^*|A)$

An alternative way of proving all three properties is simply by observing that the facility location function is non-negative, monotone, and submodular because it's a non-negative sum of non-negative, monotone and submodular terms. In our particular case, functions $f_{q,d}$ constitute such terms, hence, said properties must hold. $\square$

A.3 The double stochasticity of the dependent data forgetting objective f.

This property is a direct consequence of the (hidden) stochasticity of the facility location function. Namely, the fact that

$$\frac{1}{|D|} \cdot f_{\text{facility location}}(D') = \sum_{d \in D} \frac{1}{|D|} \max_{d' \in D'} \operatorname{sim}(d, d') = \mathbb{E}_{d \sim \mathcal{U}(D)} \max_{d' \in D'} \operatorname{sim}(d, d'),$$

where $\mathcal{U}(D)$ denotes the uniform distribution over D. Hence, the objective of the dependent data forgetting problem can be re-written as

$$f(D') = \mathbb{E}_{q \sim \mathcal{Q}} \mathbb{E}_{d \sim \mathcal{U}(q(D))} \max_{d' \in q(D')} \operatorname{sim}(d, d').$$

This beautiful observation arises a very important question: Can we write objective f as a single expectation over $(d, q) \sim \mathcal{D}$ for some distribution $\mathcal{D}$? The answer is yes. Namely, for distribution $\mathcal{D}$ with probability mass function

$$\mathbb{P}_{\mathcal{D}}(d, q) = \frac{|q(\{d\})|}{|q(D)|} \cdot \mathbb{P}_{\mathcal{Q}}(q), \quad \forall\, (d, q) \in D \times Q.$$

Note that the probability distribution $\mathcal{Q}$ is part of the input of the dependent data forgetting problem.

Proposition A.3.1. $\mathbb{P}_{\mathcal{D}}(d, q) = \frac{|q(\{d\})|}{|q(D)|} \cdot \mathbb{P}_{\mathcal{Q}}(q)$, $(d, q) \in D \times Q$ *is a probability mass function.*

Proof.

1. $\mathbb{P}_{\mathcal{D}}(d, q) \geq 0$: This is true since $\frac{|q(\{d\})|}{|q(D)|} \geq 0$ and $\mathbb{P}_{\mathcal{Q}}(q) \in [0, 1]$. Note that queries are assumed to be non-empty (i.e., $q(D) \neq \emptyset$, so no division by 0 takes place).

2. $\sum_{(d,q)\in D\times Q}\mathbb{P}_{\mathcal{D}}(d,q)=1$:

$$\begin{aligned}
\sum_{(d,q)\in D\times Q}\mathbb{P}_{\mathcal{D}}(d,q) &= \sum_{d\in D}\sum_{q\in Q}\mathbb{P}_{\mathcal{D}}(d,q)\\
&= \sum_{q\in Q}\sum_{d\in D}\mathbb{P}_{\mathcal{D}}(d,q)\\
&= \sum_{q\in Q}\sum_{d\in D}\frac{|q(\{d\})|}{|q(D)|}\cdot\mathbb{P}_{\mathcal{Q}}(q)\\
&= \sum_{q\in Q}\mathbb{P}_{\mathcal{Q}}(q)\sum_{d\in D}\frac{|q(\{d\})|}{|q(D)|}\\
&= \sum_{q\in Q}\mathbb{P}_{\mathcal{Q}}(q)\frac{\sum_{d\in D}|q(\{d\})|}{|q(D)|}\\
&\underset{\text{Lemma 3.2.2}}{=} \sum_{q\in Q}\mathbb{P}_{\mathcal{Q}}(q)\frac{|q(D)|}{|q(D)|}\\
&= \sum_{q\in Q}\mathbb{P}_{\mathcal{Q}}(q)=1.
\end{aligned}$$

□

Proposition A.3.2. *(**Proposition 3.2.4** of the main text) The objective of the dependent data forgetting problem can be written as*

$$f(D')=\mathbb{E}_{(d,q)\sim\mathcal{D}}\ f_{q,d}(D')$$

where, $\mathcal{D}$ is the distribution with probability mass function $\mathbb{P}_{\mathcal{D}}(d,q)=\frac{|q(\{d\})|}{|q(D)|}\cdot\mathbb{P}_{\mathcal{Q}}(q)$ and

$$f_{q,d}(D')=\max_{d'\in q(D')} sim(d,d').$$

Proof.

$$\begin{aligned}
f(D') &= \mathbb{E}_{q\sim\mathcal{Q}}\mathbb{E}_{d\sim\mathcal{U}(q(D))}\ f_{q,d}(D') \\
&= \sum_{q\in Q}\mathbb{P}_{\mathcal{Q}}(q) \sum_{d\in q(D)} \frac{1}{|q(D)|}\cdot f_{q,d}(D') \\
&= \sum_{q\in Q}\mathbb{P}_{\mathcal{Q}}(q) \sum_{d\in q(D)} \frac{1}{|q(D)|}\cdot f_{q,d}(D') \\
&= \sum_{q\in Q}\mathbb{P}_{\mathcal{Q}}(q) \left(\sum_{d\in q(D)} \frac{1}{|q(D)|}\cdot f_{q,d}(D') + \sum_{d\in D\setminus q(D)} 0\right) \\
&= \sum_{q\in Q}\mathbb{P}_{\mathcal{Q}}(q) \sum_{d\in D} \frac{\mathbb{I}_{\{d\in q(D)\}}(d)}{|q(D)|}\cdot f_{q,d}(D') \\
&= \sum_{q\in Q}\mathbb{P}_{\mathcal{Q}}(q) \sum_{d\in D} \frac{|q(\{d\})|}{|q(D)|}\cdot f_{q,d}(D') \\
&= \sum_{q\in Q}\sum_{d\in D}\mathbb{P}_{\mathcal{Q}}(q)\cdot \frac{|q(\{d\})|}{|q(D)|}\cdot f_{q,d}(D') \\
&= \sum_{q\in Q}\sum_{d\in D}\mathbb{P}_{\mathcal{D}}(d,q)\cdot f_{q,d}(D') \\
&= \sum_{d\in D}\sum_{q\in Q}\mathbb{P}_{\mathcal{D}}(d,q)\cdot f_{q,d}(D') \\
&= \mathbb{E}_{(d,q)\sim\mathcal{D}}\ f_{q,d}(D').
\end{aligned}$$

□

Appendix B

Supplementary material of section 3.2.3.

B.1 The gradient computations in SCG for the continuous counterpart of dependent data forgetting.

SCG builds a solution vector $\mathbf{x}^*$ starting from the origin through a series of rounds. The first step of each round t involves (approximately) computing $\nabla F(\mathbf{x}_t)$, where F is the multilinear extension of the dependent data forgetting objective f, and $\mathbf{x}_t$ is the partial solution at time t. Computing the gradient ∇F exactly is extremely expensive, however, its estimation becomes simple upon observing that f is doubly stochastic (appendix A.3).

B.1.1 Estimating $\nabla F(\mathbf{x})$

Objective $F(\mathbf{x})$ is the multilinear extension of set function $f(D') = \mathbb{E}_{(d,q)\sim\mathcal{D}}\ f_{q,d}(D')$. Nonetheless, since the multilinear extension of any set function is itself linear,

$$F(\mathbf{x}) = \mathbb{E}_{(d,q)\sim\mathcal{D}}\ F_{q,d}(\mathbf{x}),$$

where $F_{q,d}(\mathbf{x})$ is the multilinear extension of $f_{q,d}(D')$. Furthermore, ∇ is a linear operator, and hence,

$$\nabla F(\mathbf{x}) = \mathbb{E}_{(d,q)\sim\mathcal{D}}\ \nabla F_{q,d}(\mathbf{x}).$$

The standard way of approximating $\nabla F(\mathbf{x})$ is using a minibatch of size N. That is, building a sample by drawing N pairs $(d, q) \sim \mathcal{D}$, and estimating

$$\nabla F(\mathbf{x}) \approx \frac{1}{N} \sum \nabla F_{q,d}(\mathbf{x}).$$

However, this technique is very computationally expensive. Nonetheless, SCG is very efficient as it does **not** require using a minibatch to estimate ∇F. Instead, at each step $t = 1, \ldots, T$, it simply averages over the stochastic estimates of previous steps. That is, the estimated gradient of F

evaluated at the partial solution at time t (i.e., point $\mathbf{x}_t$), denoted $\mathbf{d}_t$, is computed via the recursion

$$\mathbf{d}_t = (1-\rho_t)\mathbf{d}_{t-1} + \rho_t \nabla F_{q,d}(\mathbf{x}_t),$$

where $\rho_t = \frac{4}{(t+8)^{2/3}}$, and $\mathbf{d}_0 = \mathbf{0}$. It can be shown that

$$\mathbb{E}[||\mathbf{d}_t - \nabla F(\mathbf{x}_t)||^2] \underset{t\to\infty}{\longrightarrow} 0,$$

[Mokhtari et al.(2018)]. Consequently, in our case, estimating $\nabla F(\mathbf{x}_t)$ boils down to sampling a **single** pair $(d,q) \sim \mathcal{D}$ and computing $\nabla F_{q,d}(\mathbf{x}_t)$.

B.1.1.1 Sampling $(d,q) \sim \mathcal{D}$

Due to the product rule, sampling $(d,q) \sim \mathcal{D}$ can be broken down into two simple steps:

1. Sampling a single query $q \sim \mathcal{Q}$.
2. Sampling a single datapoint $d \sim \mathcal{U}(q(D))$. In other words, sampling a single tuple d uniformly at random from answer set $q(D)$.

B.1.1.2 Computing $\nabla F_{q,d}(\mathbf{x}_t)$

It suffices computing an unbiased estimate of $\nabla F_{q,d}(\mathbf{x}_t) = \left(\frac{\partial F_{q,d}(\mathbf{x}_t)}{\partial x_1}, \ldots, \frac{\partial F_{q,d}(\mathbf{x}_t)}{\partial x_n}\right)$. Luckily, computing an unbiased estimate of $\frac{\partial F_{q,d}(\mathbf{x}_t)}{\partial x_i}$ is quite straightforward in the case of multilinear extensions [Mokhtari et al.(2018)]. An unbiased estimate of $\frac{\partial F_{q,d}(\mathbf{x}_t)}{\partial x_i}$ is given by

$$f_{q,d}(D' \cup \{d_i\}) - f_{q,d}(D' \setminus \{d_i\}), \tag{B.1}$$

where D' is a (sampled) subset of D that includes each tuple $d_i \in D$ independently with probability $(\mathbf{x}_t)_i$. Since $f_{q,d}(D') = \max_{d' \in q(D')} \text{sim}(d,d')$, equation (B.1) further simplifies. We distinguish two cases:

· **Case 1:** $d_i \notin q(D)$. In this scenario, $q(\{d_i\}) = \emptyset$. Hence (B.1) = 0.

· **Case 2:** $d_i \in q(D)$. In this scenario, $q(\{d_i\}) = \{d_i\}$. Hence,

$$\text{(B.1)} = \max\left\{0, \quad \text{sim}(d,d_i) \; - \max_{d' \in q(D)\cap D'} \text{sim}(d,d')\right\}.$$

B.2 Solving the linear program

The second step of each round t of SCG involves solving the linear program

$$\mathbf{v}_t = \arg\max_{\mathbf{v}\in\mathcal{C}}\{\langle \mathbf{d}_t, \mathbf{v}\rangle\}, \tag{B.2}$$

where, in the case of the continuous counterpart of dependent data forgetting,

$$\mathcal{C} = \{\mathbf{x} \in [0,1]^n \ : \ \sum_{i=1}^{n} x_i \leq B\}.$$

In this particular case, the solution $\mathbf{v}_t$ to the linear program is simply the binary vector with B 1s and $n - B$ 0s, where the 1s are in the position of the B largest coordinates of $\mathbf{d}_t$. Since vector $\mathbf{d}_t$ has dimension $n = |D|$, (B.2) can be solved in $\mathcal{O}(n)$ time.

REFERENCES

[Ahmed(2019)] Mohiuddin Ahmed. 2019. Data summarization: a survey. *Knowledge and Information Systems* 58 (2019), 249–273. `https://doi.org/10.1007/s10115-018-1183-0`

[Apache Hadoop(2005)] Apache Hadoop. 2005. *Hadoop*. `https://hadoop.apache.org`

[Badanidiyuru et al.(2014)] Ashwinkumar Badanidiyuru, Baharan Mirzasoleiman, Amin Karbasi, and Andreas Krause. 2014. Streaming submodular maximization: Massive data summarization on the fly. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*. 671–680.

[Barbosa et al.(2015)] Rafael Barbosa, Alina Ene, Huy Nguyen, and Justin Ward. 2015. The power of randomization: Distributed submodular maximization on massive datasets. In *International Conference on Machine Learning*. PMLR, 1236–1244.

[Calinescu et al.(2011)] Gruia Calinescu, Chandra Chekuri, Martin Pal, and Jan Vondrák. 2011. Maximizing a monotone submodular function subject to a matroid constraint. *SIAM J. Comput.* 40, 6 (2011), 1740–1766.

[Cochran(1977)] William G. Cochran. 1977. *Sampling Techniques, 3rd Edition.* John Wiley.

[Cormode(2017)] Graham Cormode. 2017. Data Sketching. *Commun. ACM* 60, 9 (aug 2017), 48–55. `https://doi.org/10.1145/3080008`

[Cormode et al.(2011)] Graham Cormode, Minos Garofalakis, Peter J Haas, Chris Jermaine, et al. 2011. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends® in Databases* 4, 1–3 (2011), 1–294.

[Davidson et al.(2022)] Susan B. Davidson, Shay Gershtein, Tova Milo, Slava Novgorodov, and May Shoshan. 2022. PHOcus: Efficiently Archiving Photos. *Proc. VLDB Endow.* 15, 12 (sep 2022), 3630–3633. `https://doi.org/10.14778/3554821.3554861`

[Dean and Ghemawat(2004)] Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified Data Processing on Large Clusters. *OSDI* (2004).

[Divjak(2022)] Ana Kuveždić Divjak. 2022. *Twinning Open Data Operational: Data preservation*. `https://todo-project.eu/en/node/552`

[EU(2016)] EU. 2016. General Data Protection Regulation (GDPR). *https://gdpr.eu/* (2016).

[Feldman(2020)] Dan Feldman. 2020. Introduction to core-sets: an updated survey. *arXiv preprint arXiv:2011.09384* (2020).

[Gershtein et al.(2020)] Shay Gershtein, Tova Milo, and Slava Novgorodov. 2020. Inventory Reduction via Maximal Coverage in E-Commerce.. In *EDBT*. 522–533.

[Hassani et al.(2017)] Hamed Hassani, Mahdi Soltanolkotabi, and Amin Karbasi. 2017. Gradient methods for submodular maximization. *Advances in Neural Information Processing Systems* 30 (2017).

[Karimi et al.(2017)] Mohammad Karimi, Mario Lucic, Hamed Hassani, and Andreas Krause. 2017. Stochastic submodular maximization: The case of coverage functions. *Advances in Neural Information Processing Systems* 30 (2017).

[Kersten(2015)] Martin Kersten. 2015. Big Data Space Fungus. In *Proceedings of the 7th CIDR*.

[Kersten(2016)] Martin Kersten. 2016. Keynote: DataFungi, from Rotting Data to Purified Information.. In *Proceedings of the 32nd ICDE*.

[Kersten and Sidirourgos(2017)] Martin Kersten and Lefteris Sidirourgos. 2017. A Database System with Amnesia. In *CIDR*.

[Krause and Golovin(2014)] Andreas Krause and Daniel Golovin. 2014. Submodular function maximization. *Tractability* 3 (2014), 71–104.

[Leskovec et al.(2007)] Jure Leskovec, Andreas Krause, Carlos Guestrin, Christos Faloutsos, Jeanne VanBriesen, and Natalie Glance. 2007. Cost-effective outbreak detection in networks. In *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. 420–429.

[Leskovec et al.(2020)] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. 2020. *Mining of massive data sets*. Cambridge university press.

[Lin and Bilmes(2012)] Hui Lin and Jeff A Bilmes. 2012. Learning mixtures of submodular shells with application to document summarization. *arXiv preprint arXiv:1210.4871* (2012).

[Liu et al.(2021)] Paul Liu, Aviad Rubinstein, Jan Vondrák, and Junyao Zhao. 2021. Cardinality constrained submodular maximization for random streams. *Advances in Neural Information Processing Systems* 34 (2021), 6491–6502.

[Milo(2019)] Tova Milo. 2019. Getting Rid of Data. *J. Data and Information Quality* 12, 1, Article 1 (nov 2019), 7 pages. `https://doi.org/10.1145/3326920`

[Minoux(1978)] Michel Minoux. 1978. Accelerated greedy algorithms for maximizing submodular set functions. In *Optimization Techniques: Proceedings of the 8th IFIP Conference on Optimization Techniques Würzburg, September 5–9, 1978*. Springer, 234–243.

[Mirzasoleiman(2017)] Baharan Mirzasoleiman. 2017. *Big data summarization using submodular functions*. Ph. D. Dissertation. ETH Zurich.

[Mirzasoleiman et al.(2015)] Baharan Mirzasoleiman, Ashwinkumar Badanidiyuru, Amin Karbasi, Jan Vondrák, and Andreas Krause. 2015. Lazier than lazy greedy. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 29.

[Mirzasoleiman et al.(2013)] Baharan Mirzasoleiman, Amin Karbasi, Rik Sarkar, and Andreas Krause. 2013. Distributed submodular maximization: Identifying representative elements in massive data. *Advances in Neural Information Processing Systems* 26 (2013).

[Mokhtari et al.(2018)] Aryan Mokhtari, Hamed Hassani, and Amin Karbasi. 2018. Conditional gradient method for stochastic submodular maximization: Closing the gap. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 1886–1895.

[Nemhauser et al.(1978)] George L Nemhauser, Laurence A Wolsey, and Marshall L Fisher. 1978. An analysis of approximations for maximizing submodular set functions—I. *Mathematical programming* 14 (1978), 265–294.

[Norouzi-Fard et al.(2018)] Ashkan Norouzi-Fard, Jakub Tarnawski, Slobodan Mitrovic, Amir Zandieh, Aidasadat Mousavifar, and Ola Svensson. 2018. Beyond 1/2-approximation for submodular maximization on massive data streams. In *International Conference on Machine Learning*. PMLR, 3829–3838.

[Orr et al.(2020)] Laurel Orr, Magdalena Balazinska, and Dan Suciu. 2020. Sample debiasing in the themis open world database system. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 257–268.

[Phillips(2017)] Jeff M Phillips. 2017. Coresets and sketches. In *Handbook of discrete and computational geometry*. Chapman and Hall/CRC, 1269–1288.

[Reinsel et al.(2018)] David Reinsel, John Gantz, and John Rydning. 2018. Data Age 2025 - The digitization of the world from edge to core. *https://www.seagate.com* (2018).

[Simon et al.(2007)] Ian Simon, Noah Snavely, and Steven M Seitz. 2007. Scene summarization for online image collections. In *2007 IEEE 11th International Conference on Computer Vision*. IEEE, 1–8.

[Tschiatschek et al.(2014)] Sebastian Tschiatschek, Rishabh K Iyer, Haochen Wei, and Jeff A Bilmes. 2014. Learning Mixtures of Submodular Functions for Image Collection Summarization. In *Advances in Neural Information Processing Systems*, Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K.Q. Weinberger (Eds.), Vol. 27. Curran Associates, Inc. `https://proceedings.neurips.cc/paper/2014/file/a8e864d04c95572d1aece099af852d0a-Paper.pdf`

[Upton and Cook(2002)] Graham J. G. Upton and Patrick Cook. 2002. A Dictionary of Statistics.

[Vondrák(2008)] Jan Vondrák. 2008. Optimal approximation for the submodular welfare problem in the value oracle model. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*. 67–74.

[Wang and Strong(1996)] Richard Y Wang and Diane M Strong. 1996. Beyond accuracy: What data quality means to data consumers. *Journal of management information systems* 12, 4 (1996), 5–33.