Making AI Forget You: Data Deletion in Machine Learning

Antonio A. Ginart[1], Melody Y. Guan[2], Gregory Valiant[2], and James Zou[3]

[1]Dept. of Electrical Engineering
[2]Dept. of Computer Science
[3]Dept. of Biomedial Data Science
Stanford University, Palo Alto, CA 94305
{tginart, mguan, valiant, jamesz}@stanford.edu

Abstract

Intense recent discussions have focused on how to provide individuals with control over when their data can and cannot be used — the EU's Right To Be Forgotten regulation is an example of this effort. In this paper we initiate a framework studying what to do when it is no longer permissible to deploy models derivative from specific user data. In particular, we formulate the problem of efficiently deleting individual data points from trained machine learning models. For many standard ML models, the only way to completely remove an individual's data is to retrain the whole model from scratch on the remaining data, which is often not computationally practical. We investigate algorithmic principles that enable efficient data deletion in ML. For the specific setting of k-means clustering, we propose two provably efficient deletion algorithms which achieve an average of over $100\times$ improvement in deletion efficiency across 6 datasets, while producing clusters of comparable statistical quality to a canonical k-means++ baseline.

1 Introduction

Recently, one of the authors received the redacted email below, informing us that an individual's data cannot be used any longer. The UK Biobank [79] is one of the most valuable collections of genetic and medical records with half a million participants. Thousands of machine learning classifiers are trained on this data, and thousands of papers have been published using this data.

```
EMAIL -- UK BIOBANK --
Subject:  UK Biobank Application [REDACTED], Participant Withdrawal Notification [REDACTED]

Dear Researcher,

As you are aware, participants are free to withdraw form the UK Biobank at any time and request that their
data no longer be used.  Since our last review, some participants involved with Application [REDACTED]
have requested that their data should longer be used.
```

The email request from the UK Biobank illustrates a fundamental challenge the broad data science and policy community is grappling with: *how should we provide individuals with flexible control over how corporations, governments, and researchers use their data?* Individuals could decide at any time that they do not wish for their personal data to be used for a particular purpose by a particular entity. This ability is sometimes legally enforced. For example, the European Union's General Data Protection Regulation (GDPR) and former Right to Be Forgotten [24, 23] both require that companies and organizations enable users to withdraw consent to their data at any time under certain circumstances. These regulations broadly affect international companies and technology platforms with EU customers and users. Legal scholars have pointed out that the continued use of AI systems directly trained on deleted data could be considered illegal under certain interpretations and ultimately concluded that: *it may be impossible to fulfill the legal aims of the Right to be Forgotten in artificial intelligence environments* [86]. Furthermore, so-called *model-inversion attacks* have demonstrated the capability of adversaries to extract user information from trained ML models [85].

33rd Conference on Neural Information Processing Systems (NeurIPS 2019), Vancouver, Canada.

Concretely, we frame the problem of data deletion in machine learning as follows. Suppose a statistical model is trained on n datapoints. For example, the model could be trained to perform disease diagnosis from data collected from n patients. To *delete* the data sampled from the i-th patient from our trained model, we would like to update it such that it becomes independent of sample i, and looks as if it had been trained on the remaining $n-1$ patients. A naive approach to satisfy the requested deletion would be to retrain the model from scratch on the data from the remaining $n-1$ patients. For many applications, this is not a tractable solution – the costs (in time, computation, and energy) for training many machine learning models can be quite high. Large scale algorithms can take weeks to train and consume large amounts of electricity and other resources. Hence, we posit that efficient data deletion is a fundamental data management operation for machine learning models and AI systems, just like in relational databases or other classical data structures.

Beyond supporting individual data rights, there are various other possible use cases in which efficient data deletion is desirable. To name a few examples, it could be used to speed-up leave-one-out-cross-validation [2], support a user data marketplace [75, 80], or identify important or valuable datapoints within a model [37].

Deletion efficiency for general learning algorithms has not been previously studied. While the desired output of a deletion operation on a *deterministic* model is fairly obvious, we have yet to even define data deletion for stochastic learning algorithms. At present, there is only a handful of learning algorithms known to support fast data deletion operations, all of which are deterministic. Even so, there is no pre-exisitng notion of how engineers should think about the asymptotic *deletion efficiency* of learning systems, nor understanding of the kinds of trade-offs such systems face.

The key components of this paper include introducing deletion efficient learning, based on an intuitive and operational notion of what it means to (efficiently) delete data from a (possibly stochastic) statistical model. We pose data deletion as an online problem, from which a notion of optimal deletion efficiency emerges from a natural lower bound on amortized computation time. We do a case-study on deletion efficient learning using the simple, yet perennial, k-means clustering problem. We propose two deletion efficient algorithms that (in certain regimes) achieve optimal deletion efficiency. Empirically, on six datasets, our methods achieve an average of over $100\times$ speedup in amortized runtime with respect to the canonical Lloyd's algorithm seeded by k-means++ [53, 5]. Simultaneously, our proposed deletion efficient algorithms perform comparably to the canonical algorithm on three different statistical metrics of clustering quality. Finally, we synthesize an algorithmic toolbox for designing deletion efficient learning systems.

We summarize our work into three contributions:

(1) We formalize the problem and notion of efficient data deletion in the context of machine learning.

(2) We propose two different deletion efficient solutions for k-means clustering that have theoretical guarantees and strong empirical results.

(3) From our theory and experiments, we synthesize four general engineering principles for designing deletion efficient learning systems.

2 Related Works

Deterministic Deletion Updates As mentioned in the introduction, efficient deletion operations are known for some canonical learning algorithms. They include linear models [55, 27, 83, 81, 18, 74], certain types of *lazy learning* [88, 6, 11] techniques such as non-parametric Nadaraya-Watson kernel regressions [61] or nearest-neighbors methods [22, 74], recursive support vector machines [19, 81], and co-occurrence based collaborative filtering [74].

Data Deletion and Data Privacy Related ideas for protecting data in machine learning — e.g. cryptography [63, 16, 14, 13, 62, 31], differential privacy [30, 21, 20, 64, 1] — do not lead to efficient data deletion, but rather attempt to make data private or non-identifiable. Algorithms that support efficient deletion do not have to be private, and algorithms that are private do not have to support efficient deletion. To see the difference between privacy and data deletion, note that every learning algorithm supports the naive data deletion operation of retraining from scratch. The algorithm is not required to satisfy any privacy guarantees. *Even an operation that outputs the entire dataset in the clear could support data deletion, whereas such an operation is certainly not private*. In this sense, the challenge of data deletion only arises in the presence of computational limitations. Privacy, on the other hand, presents statistical challenges, even in the absence of any computational limitations. With that being said, data deletion has direct connections and consequences in data privacy and security, which we explore in more detail in Appendix A.

3 Problem Formulation

We proceed by describing our setting and defining the notion of *data deletion* in the context of a machine learning algorithm and model. Our definition formalizes the intuitive goal that after a specified datapoint, x, is deleted, the resulting model is updated to be indistinguishable from a model that was trained from scratch on the dataset sans x. Once we have defined data deletion, we will conclude this section by defining a notion of *deletion efficiency* in the context of an online setting in which a stream of data deletion requests must be processed.

Throughout we denote dataset $D = \{x_1,...,x_n\}$ as a set consisting of n datapoints, with each datapoint $x_i \in \mathbf{R}^d$; for simplicity, we often represent D as a $n \times d$ real-valued matrix as well. Let A denote a (possibly randomized) algorithm that maps a dataset to a model in hypothesis space $\mathcal{H}$. We allow models to also include arbitrary metadata that is not necessarily used at inference time. Such metadata could include data structures or partial computations that can be leveraged to help with subsequent deletions. We also emphasize that algorithm A operates on datasets of any size. Since A is often stochastic, we can also treat A as implicitly defining a conditional distribution over $\mathcal{H}$ given dataset D.

Definition 3.1. Data Deletion Operation: We define a *data deletion* operation for learning algorithm A, $R_A(D,A(D),i)$, which maps the dataset D, model $A(D)$, and index $i \in \{1,...,n\}$ to some model in $\mathcal{H}$. Such an operation is a data deletion operation if, for all D and i, random variables $A(D_{-i})$ and $R_A(D,A(D),i)$ are equal in distribution, $A(D_{-i}) =_d R_A(D,A(D),i)$.

Here we focus on exact data deletion: after deleting a training point from the model, the model should be as if this training point had never been seen in the first place. The above definition can naturally be relaxed to approximate data deletion by requiring a bound on the distance (or divergence) between distributions of $A(D_{-i})$ and $R_A(D,A(D),i)$. Refer to Appendix A for more details on approximate data deletion, especially in connection to differential privacy. We defer a full discussion of this to future work.

A Computational Challenge Every learning algorithm, A, supports a trivial data deletion operation corresponding to simply retraining on the new dataset after the specified datapoint has been removed — namely running algorithm A on the dataset D_{-i}. Because of this, the challenge of data deletion is computational: **1)** Can we design a learning algorithm A, and supporting data structures, so as to allow for a computationally efficient data deletion operation? **2)** For what algorithms A is there a data deletion operation that runs in time sublinear in the size of the dataset, or at least sublinear in the time it takes to compute the original model, $A(D)$? **3)** How do restrictions on the memory-footprint of the metadata contained in $A(D)$ impact the efficiency of data deletion algorithms?

Data Deletion as an Online Problem One convenient way of concretely formulating the computational challenge of data deletion is via the lens of online algorithms [17]. Given a dataset of n datapoints, a specific training algorithm A, and its corresponding deletion operation R_A, one can consider a stream of $m \leq n$ distinct indices, $i_1,i_2,...,i_m \in \{1,...,n\}$, corresponding to the sequence of datapoints to be deleted. The online task then is to design a data deletion operation that is given the indices $\{i_j\}$ one at a time, and must output $A(D_{-\{i_1,...,i_j\}})$ upon being given index i_j. As in the extensive body of work on online algorithms, the goal is to minimize the amortized computation time. The amortized runtime in the proposed online deletion setting is a natural and meaningful way to measure deletion efficiency. A formal definition of our proposed online problem setting can be found in Appendix A.

In online data deletion, a simple lower bound on amortized runtime emerges. All (sequential) learning algorithms A run in time $\Omega(n)$ under the natural assumption that A must process each datapoint at least once. Furthermore, in the best case, A comes with a constant time deletion operation (or a deletion oracle).

Remark 3.1. *In the online setting, for n datapoints and m deletion requests we establish an asymptotic lower bound of $\Omega(\frac{n}{m})$ for the amortized computation time of any (sequential) learning algorithm.*

We refer to an algorithm achieving this lower bound as *deletion efficient*. Obtaining tight upper and lower bounds is an open question for many basic learning paradigms including ridge regression, decision tree models, and settings where A corresponds to the solution to a stochastic optimization problem. In this paper, we do a case study on k-means clustering, showing that we can achieve deletion efficiency without sacrificing statistical performance.

3.1 General Principles for Deletion Efficient Machine Learning Systems

We identify four design principles which we envision as the pillars of deletion efficient learning algorithms.

Linearity Use of linear computation allows for simple post-processing to undo the influence of a single datapoint on a set of parameters. Generally speaking, the Sherman-Morrison-Woodbury matrix identity and matrix factorization techniques can be used to derive fast and explicit formulas for updating linear models [55, 27, 83, 43]. For example, in the case of linear least squares regressions, QR factorization can be used to delete datapoints from learned weights in time $O(d^2)$ [41]. Linearity should be most effective in domains in which randomized [70], reservoir [89, 76], domain-specific [54], or pre-trained feature spaces elucidate linear relationships in the data.

Laziness Lazy learning methods delay computation until inference time [88, 11, 6], resulting in trivial deletions. One of the simplest examples of lazy learning is k-nearest neighbors [32, 4, 74], where deleting a point from the dataset at deletion time directly translates to an updated model at inference time. There is a natural affinity between lazy learning and non-parametric techniques [61, 15]. Although we did not make use of laziness for unsupervised learning in this work, pre-existing literature on kernel density estimation for clustering would be a natural starting place [44]. Laziness should be most effective in regimes when there are fewer constraints on inference time and model memory than training time or deletion time. In some sense, laziness can be interpreted as shifting computation from training to inference. As a side effect, deletion can be immensely simplified.

Modularity In the context of deletion efficient learning, modularity is the restriction of dependence of computation state or model parameters to specific partitions of the dataset. Under such a modularization, we can isolate specific modules of data processing that need to be recomputed in order to account for deletions to the dataset. Our notion of modularity is conceptually similar to its use in software design [10] and distributed computing [67]. In DC-k-means, we leverage modularity by managing the dependence between computation and data via the divide-and-conquer tree. Modularity should be most effective in regimes for which the dimension of the data is small compared to the dataset size, allowing for partitions of the dataset to capture the important structure and features.

Quantization Many models come with a sense of continuity from dataset space to model space — small changes to the dataset should result in small changes to the (distribution over the) model. In statistical and computational learning theory, this idea is known to as *stability* [60, 47, 50, 29, 77, 68]. We can leverage stability by quantizing the mapping from datasets to models (either explicitly or implicitly). Then, for a small number of deletions, such a quantized model is unlikely to change. If this can be efficiently verified at deletion time, then it can be used for fast average-case deletions. Quantization is most effective in regimes for which the number of parameters is small compared to the dataset size.

4 Deletion Efficient Clustering

Data deletion is a general challenge for machine learning. Due to its simplicity we focus on k-means clustering as a case study. Clustering is a widely used ML application, including on the UK Biobank (for example as in [33]). We propose two algorithms for deletion efficient k-means clustering. In the context of k-means, we treat the output centroids as the model from which we are interested in deleting datapoints. We summarize our proposed algorithms and state theoretical runtime complexity and statistical performance guarantees. Please refer to [32] for background concerning k-means clustering.

4.1 Quantized k-Means

We propose a quantized variant of Lloyd's algorithm as a deletion efficient solution to k-means clustering, called Q-k-means. By quantizing the centroids at each iteration, we show that the algorithm's centroids are constant with respect to deletions with high probability. Under this notion of quantized stability, we can support efficient deletion, since most deletions can be resolved without re-computing the centroids from scratch. Our proposed algorithm is distinct from other quantized versions of k-means [73], which quantize the data to minimize memory or communication costs. We present an abridged version of the algorithm here (Algorithm 1). Detailed pseudo-code for Q-k-means and its deletion operation may be found in Appendix B.

Q-k-means follows the iterative protocol as does the canonical Lloyd's algorithm (and makes use of the k-means++ initialization). There are four key differences from Lloyd's algorithm. First and foremost, the centroids are quantized in each iteration before updating the partition. The quantization maps each point to the nearest vertex of a uniform ϵ-lattice [38]. To de-bias the quantization, we apply a random phase shift to the lattice. The particulars of the quantization scheme are discussed in Appendix B. Second, at various steps throughout the computation, we *memoize* the optimization state into the model's metadata for use at deletion time (incurring an additional $O(ktd)$ memory cost). Third, we

introduce a balance correction step, which compensates for γ-imbalanced clusters by averaging current centroids with a momentum term based on the previous centroids. Explicitly, for some $\gamma \in (0,1)$, we consider any partition π_κ to be γ-imbalanced if $|\pi_\kappa| \leq \frac{\gamma n}{k}$. We may think of γ as being the ratio of the smallest cluster size to the average cluster size. Fourth, because of the quantization, the iterations are no longer guaranteed to decrease the loss, so we have an early termination if the loss increases at any iteration. Note that the algorithm terminates almost surely.

Deletion in Q-k-means is straightforward. Using the metadata saved from training time, we can verify if deleting a specific datapoint would have resulted in a different *quantized centroid* than was actually computed during training. If this is the case (or if the point to be deleted is one of randomly chosen initial centroids according to k-means++) we must retrain from scratch to satisfy the deletion request. Otherwise, we may satisfy deletion by updating our metadata to reflect the deletion of the specified datapoint, but we do not have to recompute the centroids. Q-k-means directly relies the principle of quantization to enable fast deletion in expectation. It is also worth noting that Q-k-means also leverages on the principle of linearity to recycle computation. Since centroid computation is linear in the datapoints, it is easy to determine the centroid update due to a removal at deletion time.

Algorithm 1 Quantized k-means (abridged)

Input: data matrix $D \in \mathbf{R}^{n \times d}$
Parameters: $k \in \mathbf{N}, T \in \mathbf{N}, \gamma \in (0,1), \epsilon > 0$
$c \leftarrow k^{++}(D)$ // *initialize centroids with k-means++*
Save initial centroids: $\mathsf{save}(c)$.
$L \leftarrow k$-means loss of initial partition $\pi(c)$
for $\tau = 1$ **to** T **do**
  Store current centroids: $c' \leftarrow c$
  Compute centroids: $c \leftarrow c(\pi)$
  Apply correction to γ-imbalanced partitions
  Quantize to random ϵ-lattice: $\hat{c} \leftarrow Q(c;\theta)$
  Update partition: $\pi' \leftarrow \pi(\hat{c})$
  Save state to metadata: $\mathsf{save}(c,\theta,\hat{c},|\pi'|)$
  Compute loss L'
  if $L' < L$ **then** $(c,\pi,L) \leftarrow (\hat{c},\pi',L')$ **else break**
end for
return c //output final centroids as model

Deletion Time Complexity We turn our attention to an asymptotic time complexity analysis of Q-k-means deletion operation. Q-k-means supports deletion by quantizing the centroids, so they are stable to against small perturbations (caused by deletion of a point).

Theorem 4.1. *Let D be a dataset on $[0,1]^d$ of size n. Fix parameters T, k, ϵ, and γ for Q-k-means. Then, Q-k-means supports m deletions in time $O(m^2 d^{5/2}/\epsilon)$ in expectation, with probability over the randomness in the quantization phase and k-means++ initialization.*

The proof for the theorem is given in Appendix C. The intuition is as follows. Centroids are computed by taking an average. With enough terms in an average, the effect of a small number of those terms is negligible. The removal of those terms from the average can be interpreted as a small perturbation to the centroid. If that small perturbation is on a scale far below the granularity of the quantizing ϵ-lattice, then it is unlikely to change the quantized value of the centroid. Thus, beyond stability verification, no additional computation is required for a majority of deletion requests. This result is in expectation with respect to the randomized initializations and randomized quantization phase, but is actually worst-case over all possible (normalized) dataset instances. The number of clusters k, iterations T, and cluster imbalance ratio γ are usually small constants in many applications, and are treated as such here. Interestingly, for constant m and ϵ, the expected deletion time is independent of n due to the stability probability increasing at the same rate as the problem size (see Appendix C). Deletion time for this method may not scale well in the high-dimensional setting. In the low-dimensional case, the most interesting interplay is between ϵ, n, and m. To obtain as high-quality statistical performance as possible, it would be ideal if $\epsilon \to 0$ as $n \to \infty$. In this spirit, we can parameterize $\epsilon = n^{-\beta}$ for $\beta \in (0,1)$. We will use this parameterization for theoretical analysis of the online setting in Section 3.3.

Theoretical Statistical Performance We proceed to state a theoretical guarantee on statistical performance of Q-k-means, which complements the asymptotic time complexity bound of the deletion operation. Recall that the loss for a k-means problem instance is given by the sum of squared Euclidean distance from each datapoint to its nearest centroid. Let $\mathcal{L}^*$ be the optimal loss for a particular problem instance. Achieving the optimal solution is, in general, NP-Hard [3]. Instead, we can approximate it with k-means++, which achieves $\mathbf{E}\mathcal{L}^{++} \leq (8\log k + 16)\mathcal{L}^*$ [5].

Corollary 4.1.1. *Let $\mathcal{L}$ be a random variable denoting the loss of Q-k-means on a particular problem instance of size n. Then $\mathbf{E}\mathcal{L} \leq (8\log k + 16)\mathcal{L}^* + \epsilon\sqrt{nd(8\log k + 16)\mathcal{L}^*} + \frac{1}{4}nd\epsilon^2$.*

This corollary follows from the theoretical guarantees already known to apply to Lloyd's algorithm when initialized with k-means++, given by [5]. The proof can be found in Appendix C. We can

interpret the bound by looking at the ratio of expected loss upper bounds for k-means++ and Q-k-means. If we assume our problem instance is generated by iid samples from some arbitrary non-atomic distribution, then it follows that $\mathcal{L}^* = O(n)$. Taking the loss ratio of upper bounds yields $\mathbf{E}\mathcal{L}/\mathbf{E}\mathcal{L}^{++} \leq 1 + O(d\epsilon^2 + \sqrt{d}\epsilon)$. Ensuring that $\epsilon << 1/\sqrt{d}$ implies the upper bound is as good as that of k-means++.

4.2 Divide-and-Conquer k-Means

We turn our attention to another variant of Lloyd's algorithm that also supports efficient deletion, albeit through quite different means. We refer to this algorithm as Divide-and-Conquer k-means (DC-k-means). At a high-level, DC-k-means works by partitioning the dataset into small sub-problems, solving each sub-problem as an independent k-means instance, and recursively merging the results. We present pseudo-code for DC-k-means here, and we refer the reader to Appendix B for pseudo-code of the deletion operation.

Algorithm 2 DC-k-means

Input: data matrix $D \in \mathbf{R}^{n \times d}$
Parameters: $k \in \mathbf{N}$, $T \in \mathbf{N}$, tree width $w \in \mathbf{N}$, tree height $h \in \mathbf{N}$
Initialize a w-ary tree of height h such that each node has a pointer to a dataset and centroids
for $i = 1$ **to** n **do**
  Select a leaf node uniformly at random
  node.dataset.add(D_i)
end for
for $l = h$ **down to** 0 **do**
  for each node in level l **do**
    $c \leftarrow$ k-means++(node.dataset,k,T)
    node.centroids $\leftarrow c$
    if $l > 0$ **then**
      node.parent.dataset.add(c)
    else
      save all nodes as metadata
      return c //model output
    end if
  end for
end for

DC-k-means operates on a perfect w-ary tree of height h (this could be relaxed to any rooted tree). The original dataset is *partitioned* into each leaf in the tree as a uniform multinomial random variable with datapoints as trials and leaves as outcomes. At each of these leaves, we solve for some number of centroids via k-means++. When we merge leaves into their parent node, we construct a new dataset consisting of all the centroids from each leaf. Then, we compute new centroids at the parent via another instance of k-means++. For simplicity, we keep k fixed throughout all of the sub-problems in the tree, but this could be relaxed. We make use of the tree hierarchy to *modularize* the computation's dependence on the data. At deletion time, we need only to recompute the sub-problems from *one* leaf up to the root. This observation allows us to support fast deletion operations.

Our method has close similarities to pre-existing distributed k-means algorithms [69, 67, 9, 7, 39, 8, 91], but is in fact distinct (not only in that it is modified for deletion, but also in that it operates over general rooted trees). For simplicity, we restrict our discussion to only the simplest of divide-and-conquer trees. We focus on depth-1 trees with w leaves where each leaf solves for k centroids. This requires only one merge step with a root problem size of kn/w.

Analogous to how ϵ serves as a knob to trade-off between deletion efficiency and statistical performance in Q-k-means, for DC-k-means, we imagine that w might also serve as a similar knob. For example, if $w = 1$, DC-k-means degenerates into canonical Lloyd's (as does Q-k-means as $\epsilon \to 0$). The dependence of statistical performance on tree width w is less theoretically tractable than that of Q-k-means on ϵ, but in Appendix D, we empirically show that statistical performance tends to decrease as w increases, which is perhaps somewhat expected.

As we show in our experiments, depth-1 DC-k-means demonstrates an empirically compelling trade-off between deletion time and statistical performance. There are various other potential extensions of this algorithm, such as weighting centroids based on cluster mass as they propagate up the tree or exploring the statistical performance of deeper trees.

Deletion Time Complexity For ensuing asymptotic analysis, we may consider parameterizing tree width w as $w = \Theta(n^\rho)$ for $\rho \in (0,1)$. As before, we treat k and T as small constants. Although intuitive, there are some technical minutia to account for to prove correctness and runtime for the DC-k-means deletion operation. The proof of Proposition 3.2 may be found in Appendix C.

Proposition 4.2. *Let D be a dataset on $\mathbf{R}^d$ of size n. Fix parameters T and k for DC-k-means. Let $w = \Theta(n^\rho)$ and $\rho \in (0,1)$ Then, with a depth-1, w-ary divide-and-conquer tree, DC-k-means supports m deletions in time $O(m\mathbf{max}\{n^\rho, n^{1-\rho}\}d)$ in expectation with probability over the randomness in dataset partitioning.*

4.3 Amortized Runtime Complexity in Online Deletion Setting

We state the amortized computation time for both of our algorithms in the online deletion setting defined in Section 2. We are in an asymptotic regime where the number of deletions $m = \Theta(n^\alpha)$ for $0 < \alpha < 1$ (see Appendix C for more details). Recall the $\Omega(\frac{n}{m})$ lower bound from Section 2.1. For a particular fractional power α, an algorithm achieving the optimal asymptotic lower bound on amortized computation is said to be α-*deletion efficient*. This corresponds to achieving an amortized runtime of $O(n^{1-\alpha})$. The following corollaries result from direct calculations which may be found in Appendix C. Note that Corollary 3.2.2 assumes DC-k-means is training sequentially.

Corollary 4.2.1. *With $\epsilon = \Theta(n^{-\beta})$ for $0 < \beta < 1$, Q-k-means algorithm is α-deletion efficient in expectation if $\alpha \leq \frac{1-\beta}{2}$*

Corollary 4.2.2. *With $w = \Theta(n^\rho)$ for $0 < \rho < 1$, and a depth-1 w-ary divide-and-conquer tree, DC-k-means is α-deletion efficient in expectation if $\alpha < 1 - \max\{1-\rho, \rho\}$*

5 Experiments

With a theoretical understanding in hand, we seek to empirically characterize the trade-off between runtime and performance for the proposed algorithms. In this section, we provide proof-of-concept for our algorithms by benchmarking their amortized runtimes and clustering quality on a simulated stream of online deletion requests. As a baseline, we use the canonical Lloyd's algorithm initialized by k-means++ seeding [53, 5]. Following the broader literature, we refer to this baseline simply as k-means, and refer to our two proposed methods as Q-k-means and DC-k-means.

Datasets We run our experiments on five real, publicly available datasets: `Celltype` ($N = 12{,}009$, $D = 10$, $K = 4$) [42], `Covtype` ($N = 15{,}120$, $D = 52$, $K = 7$) [12], `MNIST` ($N = 60{,}000$, $D = 784$, $K = 10$) [51], `Postures` ($N = 74{,}975$, $D = 15$, $K = 5$) [35, 34] , `Botnet` ($N = 1{,}018{,}298$, $D = 115$, $K = 11$)[56], and a synthetic dataset made from a Gaussian mixture model which we call `Gaussian` ($N = 100{,}000$, $D = 25$, $K = 5$). We refer the reader to Appendix D for more details on the datasets. All datasets come with ground-truth labels as well. Although we do not make use of the labels at learning time, we can use them to evaluate the statistical quality of the clustering methods.

Online Deletion Benchmark We simulate a stream of 1,000 deletion requests, selected uniformly at random and without replacement. An algorithm trains once, on the full dataset, and then runs its deletion operation to satisfy each request in the stream, producing an intermediate model at each request. For the canonical k-means baseline, deletions are satisfied by re-training from scratch.

Protocol To measure statistical performance, we evaluate with three metrics (see Section 4.1) that measure cluster quality. To measure deletion efficiency, we measure the wall-clock time to complete our online deletion benchmark. For both of our proposed algorithms, we always fix 10 iterations of Lloyd's, and all other parameters are selected with simple but effective heuristics (see Appendix D). This alleviates the need to tune them. To set a fair k-means baseline, when reporting runtime on the online deletion benchmark, we also fix 10 iterations of Lloyd's, but when reporting statistical performance metrics, we run until convergence. We run five replicates for each method on each dataset and include standard deviations with all our results. We refer the reader to Appendix D for more experimental details.

5.1 Statistical Performance Metrics

To evaluate clustering performance of our algorithms, the most obvious metric is the optimization loss of the k-means objective. Recall that this is the sum of square Euclidean distances from each datapoint to its nearest centroid. To thoroughly validate the statistical performance of our proposed algorithms, we additionally include two canonical clustering performance metrics.

Silhouette Coefficient [72]: This coefficient measures a type of correlation (between -1 and +1) that captures how dense each cluster is and how well-separated different clusters are. The silhouette coefficient is computed without ground-truth labels, and uses only spatial information. Higher scores indicate denser, more well-separated clusters.

Normalized Mutual Information (NMI) [87, 49]: This quantity measures the agreement of the assigned clusters to the ground-truth labels, up to permutation. NMI is upper bounded by 1, achieved by perfect assignments. Higher scores indicate better agreement between clusters and ground-truth labels.

5.2 Summary of Results

We summarize our key findings in four tables. In Tables 1-3, we report the statistical clustering performance of the 3 algorithms on each of the 6 datasets. In Table 1, we report the optimization loss ratios of our proposed methods over the k-means++ baseline.

In Table 2, we report the silhouette coefficient for the clusters. In Table 3, we report the NMI. In Table 4, we report the amortized total runtime of training and deletion for each method. **Overall, we see that the statistical clustering performance of the three methods are competitive.**

Furthermore, we find that both proposed algorithms yield orders of magnitude of speedup. As expected from the theoretical analysis, Q-k-means offers greater speedups in then the dimension is lower relative to the sample size, whereas DC-k-means is more consistent across dimensionalities.

Table 1: Loss Ratio

Dataset	k-means	Q-k-means	DC-k-means
Celltype	1.0 ± 0.0	1.158 ± 0.099	1.439 ± 0.157
Covtype	1.0 ± 0.029	1.033 ± 0.017	1.017 ± 0.031
MNIST	1.0 ± 0.002	1.11 ± 0.004	1.014 ± 0.003
Postures	1.0 ± 0.004	1.014 ± 0.015	1.034 ± 0.017
Gaussian	1.0 ± 0.014	1.019 ± 0.019	1.003 ± 0.014
Botnet	1.0 ± 0.126	1.018 ± 0.014	1.118 ± 0.102

Table 2: Silhouette Coefficients (higher is better)

Dataset	k-means	Q-k-means	DC-k-means
Celltype	0.384 ± 0.001	0.367 ± 0.048	0.422 ± 0.057
Covtype	0.238 ± 0.027	0.203 ± 0.026	0.222 ± 0.017
Gaussian	0.036 ± 0.002	0.031 ± 0.002	0.035 ± 0.001
Postures	0.107 ± 0.003	0.107 ± 0.004	0.109 ± 0.005
Gaussian	0.066 ± 0.007	0.053 ± 0.003	0.071 ± 0.004
Botnet	0.583 ± 0.042	0.639 ± 0.028	0.627 ± 0.046

Table 3: Normalized Mutual Information (higher is better)

Dataset	k-means	Q-k-means	DC-k-means
Celltype	0.36 ± 0.0	0.336 ± 0.032	0.294 ± 0.067
Covtype	0.311 ± 0.009	0.332 ± 0.024	0.335 ± 0.02
MNIST	0.494 ± 0.006	0.459 ± 0.011	0.494 ± 0.004
Gaussian	0.319 ± 0.024	0.245 ± 0.024	0.318 ± 0.024
Postures	0.163 ± 0.018	0.169 ± 0.012	0.173 ± 0.011
Botnet	0.708 ± 0.048	0.73 ± 0.015	0.705 ± 0.039

Table 4: Amortized Runtime in Online Deletion Benchmark (Train once + 1,000 Deletions)

Dataset	k-means Runtime (s)	Q-k-means Runtime (s)	Q-k-means Speedup	DC-k-means Runtime (s)	DC-k-means Speedup
Celltype	4.241 ± 0.248	0.026 ± 0.011	$163.286\times$	0.272 ± 0.007	$15.6\times$
Covtype	6.114 ± 0.216	0.454 ± 0.276	$13.464\times$	0.469 ± 0.021	$13.048\times$
MNIST	65.038 ± 1.528	29.386 ± 0.728	$2.213\times$	2.562 ± 0.056	$25.381\times$
Postures	26.616 ± 1.222	0.413 ± 0.305	$64.441\times$	1.17 ± 0.398	$22.757\times$
Gaussian	206.631 ± 67.285	0.393 ± 0.104	$525.63\times$	5.992 ± 0.269	$34.483\times$
Botnet	607.784 ± 64.687	1.04 ± 0.368	$584.416\times$	8.568 ± 0.652	$70.939\times$

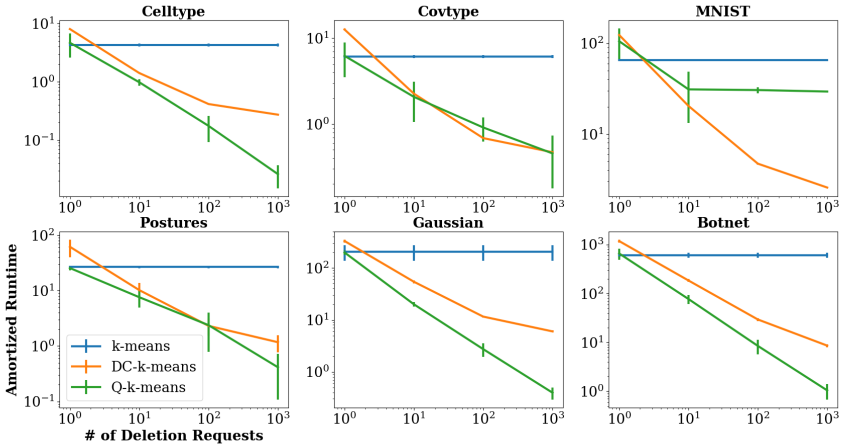

Figure 1: Online deletion efficiency: # of deletions vs. amortized runtime (secs) for 3 algorithms on 6 datasets.

In particular, note that MNIST has the highest d/n ratio of the datasets we tried, followed by Covtype, These two datasets are, respectively, the datasets for which Q-k-means offers the least speedup. On the other hand, DC-k-means offers consistently increasing speedup as n increases, despite d. Furthermore, we see that Q-k-means tends to have higher variance around its deletion efficiency, due to the randomness in centroid stabilization having a larger impact than the randomness in the dataset partitioning. We remark that 1,000 deletions is less than 10% of every dataset we test on, and statistical performance remains virtually unchanged throughout the benchmark. In Figure 1, we plot the amortized runtime on the online deletion benchmark as a function of number of deletions in the stream. We refer the reader to Appendix D for supplementary experiments providing more detail on our methods.

6 Discussion

At present, the main options for deletion efficient supervised methods are linear models, support vector machines, and non-parametric regressions. While our analysis here focuses on the concrete problem of clustering, we have proposed four design principles which we envision as the pillars of deletion efficient learning algorithms. We discuss the potential application of these methods to other supervised learning techniques.

Segmented Regression Segmented (or piece-wise) linear regression is a common relaxation of canonical regression models [58, 59, 57]. It should be possible to support a variant of segmented regression by combining Q-k-means with linear least squares regression. Each cluster could be given a separate linear model, trained only on the datapoints in said cluster. At deletion time, Q-k-means would likely keep the clusters stable, enabling a simple linear update to the model corresponding to the cluster from which the deleted point belonged.

Kernel Regression Kernel regressions in the style of random Fourier features [70] could be readily amended to support efficient deletions for large-scale supervised learning. Random features do not depend on data, and thus only the linear layer over the feature space requires updating for deletion. Furthermore, random Fourier feature methods have been shown to have affinity for quantization [90].

Decision Trees and Random Forests Quantization is also a promising approach for decision trees. By quantizing or randomizing decision tree splitting criteria (such as in [36]) it seems possible to support efficient deletion. Furthermore, random forests have a natural affinity with bagging, which naturally can be used to impose modularity.

Deep Neural Networks and Stochastic Gradient Descent A line of research has observed the robustness of neural network training robustness to quantization and pruning [84, 46, 40, 71, 25, 52]. It could be possible to leverage these techniques to quantize gradient updates during SGD-style optimization, enabling a notion of parameter stability analgous to that in Q-k-means. This would require larger batch sizes and fewer gradient steps in order to scale well. It is also possible that approximate deletion methods may be able to overcome shortcomings of exact deletion methods for large neural models.

7 Conclusion

In this work, we have devised a notion of deletion efficiency for large-scale learning systems, proposed provably deletion efficient unsupervised clustering algorithms, and identified potential algorithmic principles that may enable deletion efficiency for other learning algorithms and paradigms. So far, we have only scratched the surface of understanding deletion efficiency in learning systems. We have made the simplifying assumption that there is only one model and only one database in our system. We have also assumed that user-based deletion requests correspond to only a single data point. Understanding deletion efficiency in a system with many models and many databases, as well as complex user-to-data relationships, is an important direction for future work.

Acknowledgments: This research was partially supported by NSF Awards AF:1813049, CCF:1704417, and CCF 1763191, NIH R21 MD012867-01, NIH P30AG059307, an Office of Naval Research Young Investigator Award (N00014-18-1-2295), a seed grant from Stanford's Institute for Human-Centered AI, and the Chan-Zuckerberg Initiative. We would also like to thank I. Lemhadri, B. He, V. Bagaria, J. Thomas and anonymous reviewers for helpful discussion and feedback.

References

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[2] Y. S. Abu-Mostafa, M. Magdon-Ismail, and H.-T. Lin. *Learning from data*, volume 4. AMLBook New York, NY, USA:, 2012.

[3] D. Aloise, A. Deshpande, P. Hansen, and P. Popat. Np-hardness of euclidean sum-of-squares clustering. *Machine learning*, 75(2):245–248, 2009.

[4] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.

[5] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.

[6] C. G. Atkeson, A. W. Moore, and S. Schaal. Locally weighted learning for control. In *Lazy learning*, pages 75–113. Springer, 1997.

[7] O. Bachem, M. Lucic, and A. Krause. Distributed and provably good seedings for k-means in constant rounds. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 292–300. JMLR. org, 2017.

[8] B. Bahmani, B. Moseley, A. Vattani, R. Kumar, and S. Vassilvitskii. Scalable k-means++. *Proceedings of the VLDB Endowment*, 5(7):622–633, 2012.

[9] M.-F. F. Balcan, S. Ehrlich, and Y. Liang. Distributed k-means and k-median clustering on general topologies. In *Advances in Neural Information Processing Systems*, pages 1995–2003, 2013.

[10] O. Berman and N. Ashrafi. Optimization models for reliability of modular software systems. *IEEE Transactions on Software Engineering*, 19(11):1119–1123, 1993.

[11] M. Birattari, G. Bontempi, and H. Bersini. Lazy learning meets the recursive least squares algorithm. In *Proceedings of the 1998 Conference on Advances in Neural Information Processing Systems II*, pages 375–381, Cambridge, MA, USA, 1999. MIT Press.

[12] J. A. Blackard and D. J. Dean. Comparative accuracies of artificial neural networks and discriminant analysis in predicting forest cover types from cartographic variables. *Computers and electronics in agriculture*, 24(3):131–151, 1999.

[13] D. Bogdanov, L. Kamm, S. Laur, and V. Sokk. Implementation and evaluation of an algorithm for cryptographically private principal component analysis on genomic data. *IEEE/ACM transactions on computational biology and bioinformatics*, 15(5):1427–1432, 2018.

[14] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191. ACM, 2017.

[15] G. Bontempi, H. Bersini, and M. Birattari. The local paradigm for modeling and control: from neuro-fuzzy to lazy learning. *Fuzzy sets and systems*, 121(1):59–72, 2001.

[16] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *NDSS*, 2015.

[17] L. Bottou. Online learning and stochastic approximations. *On-line learning in neural networks*, 17(9):142, 1998.

[18] Y. Cao and J. Yang. Towards making systems forget with machine unlearning. In *2015 IEEE Symposium on Security and Privacy*, pages 463–480. IEEE, 2015.

[19] G. Cauwenberghs and T. Poggio. Incremental and decremental support vector machine learning. In *Advances in neural information processing systems*, pages 409–415, 2001.

[20] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate. Differentially private empirical risk minimization. *Journal of Machine Learning Research*, 12(Mar):1069–1109, 2011.

[21] K. Chaudhuri, A. D. Sarwate, and K. Sinha. A near-optimal algorithm for differentially-private principal components. *The Journal of Machine Learning Research*, 14(1):2905–2943, 2013.

[22] D. Coomans and D. L. Massart. Alternative k-nearest neighbour rules in supervised pattern recognition: Part 1. k-nearest neighbour classification by using alternative voting rules. *Analytica Chimica Acta*, 136:15–27, 1982.

[23] Council of European Union. Council regulation (eu) no 2012/0011, 2014. `https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:52012PC0011`.

[24] Council of European Union. Council regulation (eu) no 2016/678, 2014. `https://eur-lex.europa.eu/eli/reg/2016/679/oj`.

[25] M. Courbariaux, Y. Bengio, and J.-P. David. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

[26] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[27] R. E. W. D. A. Belsley, E. Kuh. *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*. John Wiley & Sons, Inc., New York, NY, USA, 1980.

[28] S. Dasgupta and A. Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures Algorithms*, 22(1):60–65, 2003.

[29] L. Devroye and T. Wagner. Distribution-free performance bounds for potential function rules. *IEEE Transactions on Information Theory*, 25(5):601–604, 1979.

[30] C. Dwork, A. Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014.

[31] Z. Erkin, T. Veugen, T. Toft, and R. L. Lagendijk. Generating private recommendations efficiently using homomorphic encryption and data packing. *IEEE transactions on information forensics and security*, 7(3):1053–1066, 2012.

[32] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[33] K. J. Galinsky, P.-R. Loh, S. Mallick, N. J. Patterson, and A. L. Price. Population structure of uk biobank and ancient eurasians reveals adaptation at genes influencing blood pressure. *The American Journal of Human Genetics*, 99(5):1130–1139, 2016.

[34] A. Gardner, C. A. Duncan, J. Kanno, and R. Selmic. 3d hand posture recognition from small unlabeled point sets. In *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 164–169. IEEE, 2014.

[35] A. Gardner, J. Kanno, C. A. Duncan, and R. Selmic. Measuring distance between unordered sets of different sizes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 137–143, 2014.

[36] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[37] A. Ghorbani and J. Zou. Data shapley: Equitable valuation of data for machine learning. *arXiv preprint arXiv:1904.02868*, 2019.

[38] R. M. Gray and D. L. Neuhoff. Quantization. *IEEE transactions on information theory*, 44(6):2325–2383, 1998.

[39] S. Guha, R. Rastogi, and K. Shim. Cure: an efficient clustering algorithm for large databases. In *ACM Sigmod Record*, volume 27, pages 73–84. ACM, 1998.

[40] P. Gysel, J. Pimentel, M. Motamedi, and S. Ghiasi. Ristretto: A framework for empirical study of resource-efficient inference in convolutional neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2018.

[41] S. Hammarling and C. Lucas. Updating the qr factorization and the least squares problem. 2008.

[42] X. Han, R. Wang, Y. Zhou, L. Fei, H. Sun, S. Lai, A. Saadatpour, Z. Zhou, H. Chen, F. Ye, et al. Mapping the mouse cell atlas by microwell-seq. *Cell*, 172(5):1091–1107, 2018.

[43] N. J. Higham. *Accuracy and stability of numerical algorithms*, volume 80. Siam, 2002.

[44] A. Hinneburg and H.-H. Gabriel. Denclue 2.0: Fast clustering based on kernel density estimation. In *International symposium on intelligent data analysis*, pages 70–80. Springer, 2007.

[45] W. B. Johnson and J. Lindenstrauss. Extensions of lipschitz mappings into a hilbert space. *Contemporary mathematics*, 26(189-206):1, 1984.

[46] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Computer Architecture (ISCA), 2017 ACM/IEEE 44th Annual International Symposium on*, pages 1–12. IEEE, 2017.

[47] M. Kearns and D. Ron. Algorithmic stability and sanity-check bounds for leave-one-out cross-validation. *Neural computation*, 11(6):1427–1453, 1999.

[48] A. Knoblauch. Closed-form expressions for the moments of the binomial probability distribution. *SIAM Journal on Applied Mathematics*, 69(1):197–204, 2008.

[49] Z. F. Knops, J. A. Maintz, M. A. Viergever, and J. P. Pluim. Normalized mutual information based registration using k-means clustering and shading correction. *Medical image analysis*, 10(3):432–439, 2006.

[50] S. Kutin and P. Niyogi. Almost-everywhere algorithmic stability and generalization error: Tech. rep. Technical report, TR-2002-03: University of Chicago, Computer Science Department, 2002.

[51] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[52] D. Lin, S. Talathi, and S. Annapureddy. Fixed point quantization of deep convolutional networks. In *International Conference on Machine Learning*, pages 2849–2858, 2016.

[53] S. Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[54] D. G. Lowe et al. Object recognition from local scale-invariant features. In *iccv*, volume 99, pages 1150–1157, 1999.

[55] J. H. Maindonald. *Statistical Computation*. John Wiley & Sons, Inc., New York, NY, USA, 1984.

[56] Y. Meidan, M. Bohadana, Y. Mathov, Y. Mirsky, A. Shabtai, D. Breitenbacher, and Y. Elovici. N-baiot—network-based detection of iot botnet attacks using deep autoencoders. *IEEE Pervasive Computing*, 17(3):12–22, 2018.

[57] V. M. Muggeo. Estimating regression models with unknown break-points. *Statistics in medicine*, 22(19):3055–3071, 2003.

[58] V. M. Muggeo. Testing with a nuisance parameter present only under the alternative: a score-based approach with application to segmented modelling. *Journal of Statistical Computation and Simulation*, 86(15):3059–3067, 2016.

[59] V. M. Muggeo et al. Segmented: an r package to fit regression models with broken-line relationships. *R news*, 8(1):20–25, 2008.

[60] S. Mukherjee, P. Niyogi, T. Poggio, and R. Rifkin. Learning theory: stability is sufficient for generalization and necessary and sufficient for consistency of empirical risk minimization. *Advances in Computational Mathematics*, 25(1-3):161–193, 2006.

[61] E. A. Nadaraya. On estimating regression. *Theory of Probability & Its Applications*, 9(1):141–142, 1964.

[62] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 334–348. IEEE, 2013.

[63] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa. Oblivious multi-party machine learning on trusted processors. In *USENIX Security Symposium*, pages 619–636, 2016.

[64] N. Papernot, M. Abadi, U. Erlingsson, I. Goodfellow, and K. Talwar. Semi-supervised knowledge transfer for deep learning from private training data. *arXiv preprint arXiv:1610.05755*, 2016.

[65] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[66] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[67] D. Peleg. Distributed computing. *SIAM Monographs on discrete mathematics and applications*, 5:1–1, 2000.

[68] T. Poggio, R. Rifkin, S. Mukherjee, and P. Niyogi. General conditions for predictivity in learning theory. *Nature*, 428(6981):419, 2004.

[69] J. Qin, W. Fu, H. Gao, and W. X. Zheng. Distributed k-means algorithm and fuzzy c-means algorithm for sensor networks based on multiagent consensus theory. *IEEE transactions on cybernetics*, 47(3):772–783, 2016.

[70] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *Advances in neural information processing systems*, pages 1177–1184, 2008.

[71] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[72] P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[73] V. Schellekens and L. Jacques. Quantized compressive k-means. *IEEE Signal Processing Letters*, 25(8):1211–1215, 2018.

[74] S. Schelter. "amnesia"–towards machine learning models that can forget user data very fast. In *1st International Workshop on Applied AI for Database Systems and Applications (AIDB'19)*, 2019.

[75] F. Schomm, F. Stahl, and G. Vossen. Marketplaces for data: an initial survey. *ACM SIGMOD Record*, 42(1):15–26, 2013.

[76] B. Schrauwen, D. Verstraeten, and J. Van Campenhout. An overview of reservoir computing: theory, applications and implementations. In *Proceedings of the 15th european symposium on artificial neural networks. p. 471-482 2007*, pages 471–482, 2007.

[77] S. Shalev-Shwartz, O. Shamir, N. Srebro, and K. Sridharan. Learnability, stability and uniform convergence. *Journal of Machine Learning Research*, 11(Oct):2635–2670, 2010.

[78] C. E. Shannon. Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715, 1949.

[79] C. Sudlow, J. Gallacher, N. Allen, V. Beral, P. Burton, J. Danesh, P. Downey, P. Elliott, J. Green, M. Landray, et al. Uk biobank: an open access resource for identifying the causes of a wide range of complex diseases of middle and old age. *PLoS medicine*, 12(3):e1001779, 2015.

[80] H.-L. Truong, M. Comerio, F. De Paoli, G. Gangadharan, and S. Dustdar. Data contracts for cloud-based data marketplaces. *International Journal of Computational Science and Engineering*, 7(4):280–295, 2012.

[81] C.-H. Tsai, C.-Y. Lin, and C.-J. Lin. Incremental and decremental training for linear classification. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 343–352. ACM, 2014.

[82] S. Van Der Walt, S. C. Colbert, and G. Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

[83] C. F. Van Loan and G. H. Golub. *Matrix computations*. Johns Hopkins University Press, 1983.

[84] V. Vanhoucke, A. Senior, and M. Z. Mao. Improving the speed of neural networks on cpus. Citeseer.

[85] M. Veale, R. Binns, and L. Edwards. Algorithms that remember: model inversion attacks and data protection law. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 376(2133):20180083, 2018.

[86] E. F. Villaronga, P. Kieseberg, and T. Li. Humans forget, machines remember: Artificial intelligence and the right to be forgotten. *Computer Law & Security Review*, 34(2):304–313, 2018.

[87] N. X. Vinh, J. Epps, and J. Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *Journal of Machine Learning Research*, 11(Oct):2837–2854, 2010.

[88] G. I. Webb. *Lazy Learning*, pages 571–572. Springer US, Boston, MA, 2010.

[89] J. Yin and Y. Meng. Self-organizing reservior computing with dynamically regulated cortical neural networks. In *The 2012 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2012.

[90] J. Zhang, A. May, T. Dao, and C. Ré. Low-precision random fourier features for memory-constrained kernel approximation. *arXiv preprint arXiv:1811.00155*, 2018.

[91] W. Zhao, H. Ma, and Q. He. Parallel k-means clustering based on mapreduce. In *IEEE International Conference on Cloud Computing*, pages 674–679. Springer, 2009.