Reproducibility Challenge NeurIPS 2019 Report

Debanuj Nayak
Indian Institute of Technology Gandhinagar
Gandhinagar, Gujarat, India
debanuj.nayak@iitgn.ac.in

Ritik Dutta
Indian Institute of Technology Gandhinagar
Gandhinagar, Gujarat, India
dutta.ritik@iitgn.ac.in

Abstract

In this report we present our results on reproducing the paper titled "Making AI Forget You: Data Deletion in Machine Learning" by Ginart et al. We chose the replication track for our reproducibility task. We were able to successfully reproduce the results mentioned in the paper. Our experiments support the claim made by the authors.

1 Overview

The theme of the paper by Ginart et al. is to formulate the problem of efficiently deleting individual data points from trained machine learning models and propose methods to solve this problem. The authors propose two provably efficient data deletion algorithms which they claim achieve an average of over 100x improvement in deletion efficiency across 6 datasets, while also producing clusters of comparable statistical quality to a canonical k-means++ baseline.

The authors frame the problem of data deletion as follows: consider a statistical model that is trained on n datapoints. Upon receiving a request to delete the i^{th} data point, we would like to update our trained model such that it is independent of the i^{th} sample, and looks as if it was trained on the remaining $n-1$ points.

A naive way to approach this problem would be to delete the data point from the dataset and re-train the model from scratch. However, the time taken by this method would be very high.

The authors summarise their work into the following three contributions:

1. They formalise the problem and the notion of efficient data deletion from a (possibly stochastic) statistical model.
2. They propose two different deletion efficient solutions for the k-means clustering that have theoretical guarantees and strong empirical results.
3. They also synthesize four general engineering principles for designing deletion efficient learning systems.

This paper presents two novel deletion efficient algorithms for k-means: Quantized k-means (Q k-means) and Divide and Conquer k-means (DC k-means).

2 Related Work

The authors mention some known efficient deletion operations known for some canonical learning algorithms, which fall under the category of deterministic deletion. They also draw distinctions between the somewhat similar but different idea of protecting data in the domains of cryptography and differential privacy, where the main objective is to make the data private or non-identifiable, and not delete them efficiently.

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

3 Task

As a part of the replication track, our primary task was to implement the two proposed algorithms and the baseline method (k-means with k-means++ initialisation) mentioned in the paper.

The next step was to compare the deletion efficiency and clustering performance of these two algorithms with the baseline. For the baseline, the deletion procedure was equivalent to retraining the model from scratch.

4 Deletion-Efficient Algorithms

4.1 Q k-means

Q k-means uses the technique of quantization. In this algorithm, the centers are quantized, i.e, rounded off to the nearest point on an integer grid spanning the whole feature space. This happens after each iteration of k-means, and is stored as metadata.
As long as the clusters are well balanced, deletion of some data points generally will not change the actual center much. Thus the nearest point on the grid stays same, and hence the quantized centers remain same. In such cases we don't need to make an changes to our model.
However, if the quantized centers change at any step, which can be verified from the metadata, we need to retrain from scratch.
Q k-means is designed in such a way that the second scenario happens very less, and most of the times, we require no change. Thus while amortizing, the average time for each deletion is very less.

4.2 DC k-means

DC k-means stands for Divide and Conquer k-means. The algorithm initializes a tree, and distributes each data point to one of its leaves. Small instances of k-means run inside each leaf, and the centers found are then added into the parent node's data. This process continues until we reach the root. The centers returned from the k-means instance running in the root are returned as final centers.
Now whenever a data point is deleted, it only affects one leaf of the tree. The algorithm then just keeps on computing the centers along the path from that leaf to the root, as every other node not in this path stays intact.
This decrease in computation leads to a decreased time for each deletion.

5 Implementation

We were able to successfully implement both the proposed algorithms.

First, we implemented our own version of k-means class in Python using Numpy. The data deletion operation was added, which would just remove the given data point from the data and run the original algorithm on the new dataset.

The Q k-means class and DC k-means class are built on top of the k-means class.

The description of the algorithms in the paper was sufficiently clear, and we were able to implement them by reading the paper thoroughly. The appendix provided at the end of the paper was also very helpful in properly understanding the algorithms.

The implementation can be found here:
https://github.com/DebanujNayak/Reproducibility-Neurips-Code

6 Experiment Methodology

Our experiment methodology is similar to the one mentioned in the paper. The difference is that the results for deletion efficiency and cluster performance are not calculated together. We calculate these separately. This process is then applied for all the three algorithms on some of the datasets that were used by authors.
These datasets are

- Celltype $n = 12009, d = 10, k = 4$
- Covtype $n = 15129, d = 42, k = 7$
- Postures $n = 74975, d = 15, k = 5$
- MNIST $n = 60000, d = 784, k = 10$
- Botnet $n = 1018298, d = 115, k = 11$

Amortized time

1. The training algorithm would first train on the entire dataset. Time for training will be recorded.
2. Then, a uniform random sample of 1000 data points will be selected without replacement from the original dataset, and each method must satisfy these 1000 deletions. Time taken for each deletion is also recorded.
3. The sum of the training time and the time taken for 1000 deletions is calculated. Let's call it total computation time. The amortized time is then calculated by dividing the total computation time by 1000.
4. The amortized are reported for each of the three methods (baseline, Q k-means, DC k-means) are reported.
5. In all of these steps, the number of iterations of the k means step was always set to 10.
6. The experiments we did were for Covtype and Celltype.

Clustering Performance

1. The three methods (baseline, Q k-means, DC k-means) would be run on all the six datasets.
2. For each method and dataset, we would report the k means loss, silhouette coefficient and Normalized Mutual information which are measures of clustering performance.
3. In all of these steps, the inbuilt k-means step set to 10 iterations.
4. Done for all five datasets mentioned.

The authors ran their experiments on an Intel Xeon E5-2640v4 processor. Since we wanted to compare our results with the authors, we ran our experiments on a personal computer which uses an intel core i7 processor, instead of using any GPU accelerated platform.

7 Results and Discussions

7.1 Amortized Running Times

Table 1: Results for Celltype

Algorithm	Amortized Time (authors) (s)	Amortized Time (our) (s)
k-means	4.241	2.183691
Q k-means	0.026	0.002281
DC k-means	0.272	0.270261

Our result is very similar to what the authors reported and follow the same trend as that of the authors. Q k-means is the fastest one, DC k-means performs decently whereas k Means is the slowest. Our k-means is slightly faster than the authors Our Q k-means does the job around 10 times faster than the authors. DC k-means takes nearly the same amount of time.

The primary reason we could not show results for the rest of the datasets, because the 1000 deletions takes a lot of time to run for the k-means baseline . Moreover these computations can't be shifted to a faster machine, as this would make the comparison pointless as the results provided the authors are time measurements from a single core 2.4 GHz machine.

Table 2: Results for Covtype

Algorithm	Amortized Time (authors) (s)	Amortized Time (our) (s)
k-means	6.114	3.395472171
Q k-means	0.454	1.596392673
DC k-means	0.469	0.417841124

7.2 Clustering Performance

7.2.1 Loss

The average loss ratio values for Q k-means and DC k-means are very close to one, and follow the average loss reported by the authors in the paper. We have also reported the actual mean and standard deviation for each of three algorithms on he five datasets. The standard deviation for Q k-means on MNIST and DC kmeans on Botnet are a bit more, than the other algorithms. This could have been due to presence of bad initialization among the five trials.

Table 3: Average Loss ratio w.r.t. k-means

Dataset	k-means	Q k-means	DC k-means
Celltype	1.00	1.06467	1.22308
Covtype	1.00	1.01619	1.01277
Postures	1.00	1.00377	1.00557
MNIST	1.00	1.11655	1.00906
Botnet	1.00	1.09184	1.15093

Table 4: Average Loss

Dataset	k-means	Q k-means	DC k-means
Celltype	193.004	205.487	236.062
Covtype	15099.914	15344.511	15292.779
Postures	15669.539	15728.770	15756.832
MNIST	2362316.279	1637645.787	2383728.595
Botnet	187342.773	204548.678	215618.555

Table 5: Standard Deviation of Loss

Dataset	k-means	Q k-means	DC k-means
Celltype	5.625	7.555	30.054
Covtype	459.104	457.761	356.201
Postures	132.635	79.518	72.920
MNIST	7147.371	17707.6789	7191.106436
Botnet	17014.840	588.240	22098.862

7.2.2 Silhouette Coefficient and Normalized Mutual Information

The Silhouette Coefficient and Normalized mutual Information (NMI) calculated for each algorithm and each dataset from our experiments are very close to the values reported by the authors in the paper. Overall, this shows that the values reported by the authors are correct and these can be successfully reproduced.

Table 6: Silhouette Coefficient

Dataset	k-means	Q k-means	DC k-means
Celltype	0.3802 ± 0.0152	0.3120 ± 0.0363	0.5047 ± 0.0507
Covtype	0.2266 ± 0.0388	0.2096 ± 0.0129	0.2140 ± 0.0169
Postures	0.1050 ± 0.0073	0.1000 ± 0.0043	0.1049 ± 0.0033
MNIST	0.0624 ± 0.0067	0.0557 ± 0.0047	0.0725 ± 0.0046
Botnet	0.5887 ± 0.0380	0.6065 ± 0.0296	0.6111 ± 0.0437

Table 7: Normalized Mutual Information

Dataset	k-means	Q k-means	DC k-means
Celltype	0.3778 ± 0.0399	0.3744 ± 0.0499	0.2217 ± 0.0239
Covtype	0.3147 ± 0.0170	0.3107 ± 0.0136	0.3251 ± 0.0308
Postures	0.1647 ± 0.0202	0.1590 ± 0.0127	0.1732 ± 0.0086
MNIST	0.4787 ± 0.0250	0.4630 ± 0.0083	0.4870 ± 0.0125
Botnet	0.7007 ± 0.0323	0.7058 ± 0.0375	0.6835 ± 0.0304