COMP551 - Project 4

Christopher Harper, Suzuka Kokubu, Evan Savage

December 14th, 2019

Abstract

As public concern over data privacy mounts, models which are able to delete data points and be retrained efficiently become more necessary. This paper investigates the results observed in "Making AI Forget You: Data Deletion in Machine Learning"[1], identifying improved baselines for K-means, and the implementation using hierarchical density-based spatial clustering of applications with noise (HDBSCAN). The proposed baselines in the original article are poorly optimized, and significant improvements are attainable. HDBSCAN deletion is a new experimental approach that outpaces a number of the K-means baselines without being fully optimized and without the need to retrain from scratch. With more time to spend on the novel approach, more accurate and less time-intensive results could be produced.

1 Introduction

The security and privacy of personal data has become a major concern among the general public over the last few years. Machine learning has made great use of personal data, but concerns over privacy may prompt data contributors to withdraw permission. Most models do not have an efficient method of dealing with this and usually require complete retraining, which can be costly in terms of time and computational resources. The purpose of this paper is to validate the findings of the previously published paper, "Making AI Forget You: Data Deletion in Machine Learning"[1], which explores deletion efficient algorithms. In particular, this paper explores deletion efficient clustering algorithms.

Unsupervised learning is a problem setting where data is unlabelled, and the intent is typically to group or assess the relationships between features[5]. Clustering is one of the most common techniques, and is used to assign data points to clusters. K-means clustering uses the idea of centroids to group split data into k separate groups (where k is a hyperparameter), that minimizes the sum of distances between each point and its nearest centroid. A visualization of the results of k-mean clustering in both 2 and 3 dimensions can be seen in Figure 1.

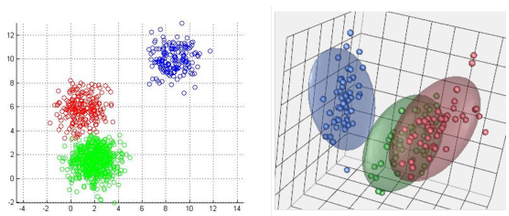

Figure 1: Results of K-means clustering in two and three dimensions [5]

Quantized-K-Means Clustering (Q-K-Means) is essentially the same as standard K-Means, but quantizes clusters prior to partitioning. A notable consequence of this assumption is that we can assume the response to deletion of each cluster is stable in expectation. The result of this is that a properly quantized centroid is unlikely to move due to a deletion, which is enormously advantageous in this problem setting. [1]

Divide and Conquer-K-Means Clustering (DC-K-Means) works by subdividing the dataset into smaller sub-problems; the smaller sub-problems are solved as independent k-means problems, and recursively merging the results. Typically, this method offers improved run-time over standard K-means, particularly when there are a large number of clusters. [1]

K-means++ is a convenient initialization algorithm that is used throughout this paper. [6] It selects one datapoint at random as an initial center, and computes the distance to all other datapoints. A second center is selected from the dataset with probability proportional to the distance (most frequently, euclidian) from the initial center. This process is repeated until k centroids have been initialized. This method produces dramatic improvements in the accuracy of k-means for a fixed number of iterations. Despite the computational overhead, it is usually faster to achieve the same level of accuracy as a randomly initialized algorithm.

Hierarchical density-based spatial clustering of applications with noise (HDBSCAN) is another clustering algorithm, and was not researched in the original paper. HDBSCAN is an extension of the popular clustering algorithm DBSCAN, implemented by researchers Campello, Moulavi, and Sander [2]. The original DBSCAN algorithm [3] works by initially finding points within a user-selected hyperparameter euclidian distance from one another. If there are enough neighboring points within this distance, then the point under consideration is the core point of a cluster. If a point does not have enough neighbors to be a core point, but at least one neighbor is a core point, then the point under consideration is a border point belonging to the same cluster as the neighboring core points. If a point is neither a core or border point, it is labelled as noise and does not belong to any cluster. DBSCAN also does not require the the user to select an amount of clusters to emerge from the dataset, contrary to K-means, which can be positive or negative depending upon what all is known about the dataset.

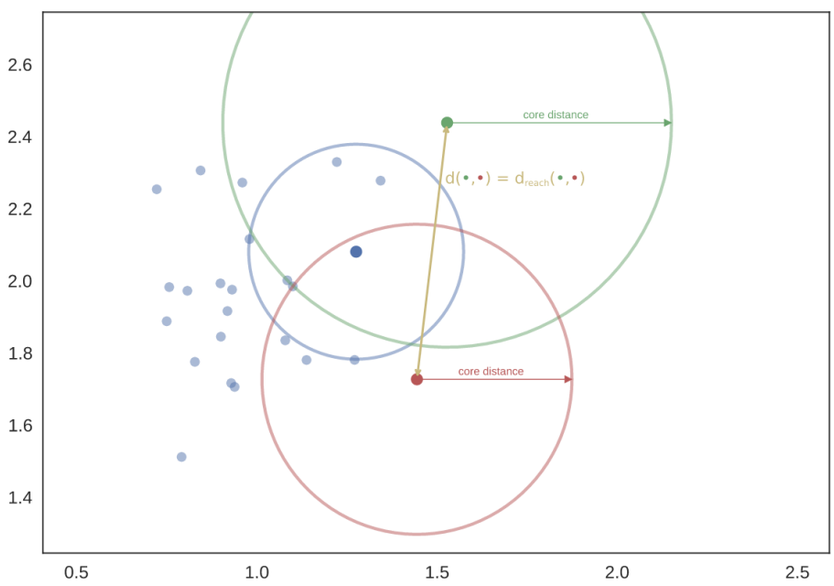

Figure 2: Visualization of mutual reachability distance (MRD) calculation[4]

DBSCAN only works well for datasets with uniform densities, since neighboring points are only considered within an initial euclidian distance that must be selected by the user. HDBSCAN strives to alleviate this concern by introducing a hierarchical structure to consider differing densities within. Every point in the dataset is checked for j closest neighbors, and the distance to the j^{th} closest neighbor is considered as a contender for the "mutual reachability distance" (euclidian) from any one point in the dataset to another (Figure 2). The "mutual reachability distance" between two points is calculated as:

$$d_{mreach-j}(a,b) = max\{core_j(a), core_j(b), d(a,b)\}$$

where $d(a,b)$ is representative of the distance between a and b [**?**]. It would take a very long time to brute-force calculate all of the MRDs (mutual reachability distance), but thanks to graph theory, HDBSCAN creates a minimum spanning tree to efficiently calculate a graph representative of the varying densities in the given dataset where every edge is representative

of the MRD between the two connected points. HDBSCAN then removes MRD edges above a certain threshold, and the clusters of points begin to emerge. Any leafs with an edge above the MRD threshold are categorized as noise. The one parameter that a user needs to specify for HDBSCAN is the minimum amount of points needed to make a cluster. This selection directly correlates to the amount of possible clusters that can emerge from the dataset and differs from K-means in this respect similar to DBSCAN.

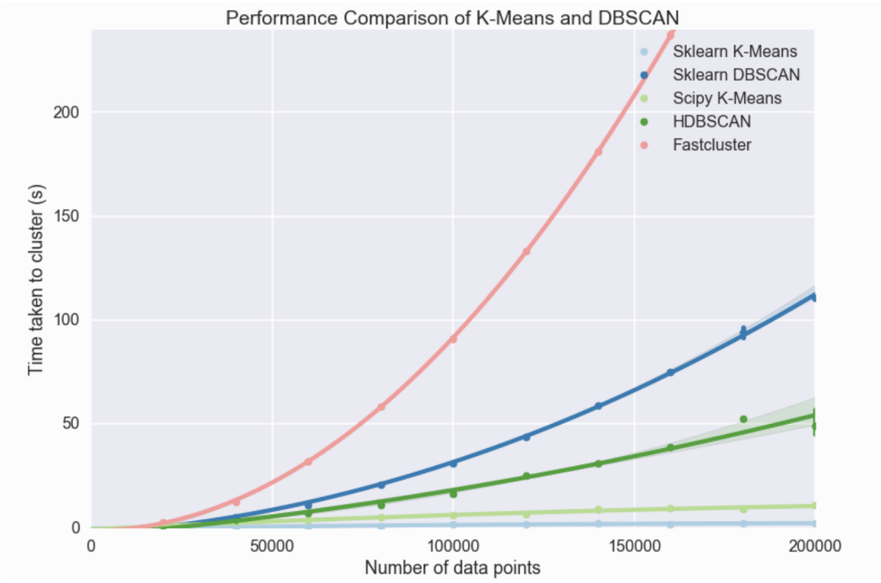

Figure 3: Performance comparisons of highly-efficient clustering algorithms.[3]

When surveying other algorithms to explore for deletion efficiency, DBSCAN and HDBSCAN emerged as contenders since their training efficiency is closets to that of K-means (Figure 3). No currently-known clustering algorithm is faster than k-means at training, though it's efficiency does not always mean it is the best fit. HDBSCAN was explored as another approach for a deletion method that could be free of ever needing to retrain from scratch. Even with a higher cost for training, the assumption was made that it could later catch up based on a lower cost deletion method.

2 Relevant Works

"Towards Making Systems Forget with Machine Unlearning"[7] is an earlier work outlining the need for "machine unlearning", or simply "unlearning". It provides a general approach that expresses models in terms of summations, and restricting the impact of unlearning to specific terms in that summation. This approach is restricted to specific models which admit a summation representation (beyond the scope of this paper), which includes naive Bayes classifiers, support vector machines, and k-means clustering. An important result of this paper is that the summation representation and retraining is asymptotically faster than naive training.

"Certified Data Removal from Machine Learning Models"[8] proposes the concept of "certified removal" and provides a theoretical guarantee that a model with datum removed is identical to a model that never observed those datum, particularly in the context of linear classification. The results show that linear models can unlearn more efficiently than naive retraining, but requires inversion of the Hessian (which may be prohibitive), and it is not compatible non-convex loss functions.

"Machine Unlearning"[9] uses sharded, isolated, sliced and aggregated (SISA) training to facilitate rapid retraining. Training data is split into "shards", and models are trained on shards independently prior to aggregation. When a deletion request is received, only the shards containing that data point require retraining. Further, shards may be subdivided into "slices", with each slice being presented sequentially to the model. The resultant parameters from each additional slice can be stored, which allows the model to be retrained and initialized using the parameters that were found prior to the addition of the slice of interest. In contrast

to naive retraining, this method improved run-time by a factor of 3.3 and 1.658 using the Purchase and SVHN datasets respectively, with 0.003% of the dataset being forgotten.

3 Methodology

3.1 Improving K-means Baselines

It is stated explicitly in the original paper that the baseline was not well optimized. Further, the authors used their own code which was found to run more slowly with comparable accuracy to the KMeans clustering algorithm provided by scikit learn. The convergence tolerance provided in the authors implementation is an order of magnitude larger than is typical for KMeans. Using this same tolerance, and restricting the number of initializations to one produced a 100x improvement in run-time, without sacrificing the statistical performance. We also have implemented KMeans algorithm to compare the performance with the algorithm provided by scikit learn, while original implementations of Q-Kmeans and DC-Kmeans were used. We have followed the same manner as the paper: standard deviation estimates was obtained by running five replicates, scikit learn library was used to calculate silhouette coefficients and normalized mutual information, and 10000 samples were randomly sampled to calculate silhouette coefficients due to expensive computation.

The methods used to compare statistical clustering performance are silhouette coefficient and normalized mutual information. The silhouette coefficient measures similarity of an object to own cluster with a comparison to other clusters with range of -1 to 1, where the similarity is indicated by higher values. The normalized mutual information measures the mutual dependency between two random samples with range of 0 to 1, and the higher dependency is indicated by higher values.

Studies on improving results of KMeans mentions that simply repeating the algorithm with different initialization of centroids leads to increase of accuracy.[10] Therefore, experiments were repeated multiple times to obtain better results. All the experiments were accomplished using Google Colaboratory.

Datasets used for the baseline improvements were Celltypes and Covtype. Celltypes consists of 12,009 single cell RNA sequences from mixture of 4 cell types with 10 dimensions. Covtype consists of 15120 samples of 52 cartographic variables such as elevation and hill shade at various times of day for 7 forest cover types.[1]

3.2 Implementing HDBSCAN Deletion Algorithm

HDBSCAN can be downloaded as an extension to the sklearn toolkit in python, and all training for any of the datasets under consideration was handled by this module. As an input parameter to the model, the user can specify whether or not to generate the minimum spanning tree (MST) during training as a data structure for later reference. The MST was generated since it is used as the core consideration of how to update the model after a deletion. It was converted to a 'networkx' graph for computational efficiency of existing node paths and edge checks. Both the original dataset and output labels for each datapoint were also converted to dictionaries for efficient referencing and updates.

When randomly choosing a node to delete, it was always checked first whether it was a branch or a leaf. Any leaf in the MST was swiftly able to be deleted, since only one edge would be affected. If a node was found to be a branch, after its deletion, new edges needed to be created. One less than the number of outgoing edges from the deleted node would need to be created to meet the original constraints of a MST. An approximate MRD between each of the nodes connected to the deleted node was created since they would no longer be connected without the deleted node. The max MRD among the connected nodes was then used as a radius to sweep around said nodes to create a list of nodes that could be possibly connected to reconstruct the MST.

Next, an adjacency matrix was constructed from the list of possible connections to see which nodes were not already connected to one another. If two nodes were found to be already connected, they were not considered in the next step because adding another edge between them would break the constraints of a MST. Then, the deletion node was deleted, and all possible connections in the adjacency matrix were checked again to see if the removal of the node resulted in a disconnect. The shortest approximate MRD between the nodes

under consideration was constructed as a new edge (with MRD weight) for the MST, and the process was repeated until a full MST was formed again from the deletion of the original node.

The final check for HDBSCAN deletion is in respect to what label the deleted node had. If the deleted node was labelled as noise, then no updates needed to happen to any surrounding nodes. Otherwise, surrounding label updates were possibly needed if the deletion of any node resulted in its respective cluster dropping under the minimum points required to make a cluster. To check this, all nodes originally connected to the deleted node were checked for neighbors with labels equal to that of the deleted node. If any of the neighboring counts reached the minimum point threshold, then no label changes were needed. Otherwise, the labels were changed to noise since they did not meet the minimum threshold for a cluster and would not have originally either. After these checks are complete, another deletion can commence.

4 Results

4.1 K-means Baselines

The original paper compares statistical clustering performance of the KMeans, Q-KMeans, and DC-KMeans. We have reproduced this results of 2 datasets using original implementations of these Q-KMeans and DC-KMeans as well as two different implementations of KMeans; our own implementation (k-means*) and the algorithm provided by scikit learn (sk-k-means). The table 1 shows that the silhouette coefficient for the clusters, and the table 2 report the normalized mutual information. The values inside parentheses show the values reported by the paper. We were able to obtain higher silhouette coefficients for the algorithm from scikit learn, Q-KMeans, and DC-KMeans. Also, we were able to obtain higher normalized mutual information for our implementation of KMeans, the algorithm from scikit learn and Q-Kmeans with Covtype dataset, and DC-Kmeans with Celltype dataset.

Table 1: Silhouette Coefficients (higher is better)

Dataset	k-means*	sk-k-means	Q-k-means	DC-k-means
Celltype	0.373 ± 0.024 (0.384 ± 0.001)	0.385 ± 0.0 (0.384 ± 0.001)	0.383 ± 0.012 (0.367 ± 0.048)	0.444 ± 0.036 (0.422 ± 0.057)
Covtype	0.217 ± 0.023 (0.238 ± 0.027)	0.282 ± 0.007 (0.238 ± 0.027)	0.249 ± 0.035 (0.203 ± 0.026)	0.250 ± 0.031 (0.222 ± 0.017)

Table 2: Normalized Mutual Information (higher is better)

Dataset	k-means*	sk-k-means	Q-k-means	DC-k-means
Celltype	0.370 ± 0.02 (0.36 ± 0.0)	0.36 ± 0.0 (0.36 ± 0.0)	0.363 ± 0.021 (0.336 ± 0.032)	0.345 ± 0.008 (0.294 ± 0.067)
Covtype	0.345 ± 0.009 (0.311 ± 0.009)	0.326 ± 0.008 (0.311 ± 0.009)	0.338 ± 0.006 (0.332 ± 0.024)	0.28 ± 0.011 (0.335 ± 0.02)

Table 3 shows that amortized runtime for KMeans, Q-KMeans, and DC-KMeans. The runtimes were computed with 1000 samples of deletions for three methods. However, due to expensive time complexity, only 20 samples were deleted on our implementation of KMeans(k-means*). We were able to obtain faster runtimes for all methods except the our implementation of KMeans.

Table 3: Amortized Runtime in Online Deletion Benchmark (Train once + 1000 (*or 20) Deletions)

Dataset	k-means*	sk-k-means	Q-k-means	DC-k-means	HDBSCAN
Celltype	43.871 ± 0.242 (4.241 ± 0.248)	0.046 ± 0.042 (4.241 ± 0.248)	0.007 ± 0.005 (0.026 ± 0.011)	0.271 ± 0.002 (0.272 ± 0.007)	0.5 ± 0.17
Covtype	60.527 ± 0.456 (6.114 ± 0.216)	0.124 ± 0.127 (6.114 ± 0.216)	0.428 ± 0.277 (0.454 ± 0.276)	0.406 ± 0.017 (0.469 ± 0.021)	0.783 ± 0.22

4.2 HDBSCAN Deletion

On the two datasets under consideration in this report, HDBSCAN deletions outperformed k-means deletions from the original paper, but it did not quite meet the thresholds of Q-k-means and DC-k-means (Table 3). This approach was highly experimental and took much time to implement from scratch, so with more time available on researching this approach, it could turn out to be a viable deletion method moving forward. Some of the checks in the algorithm were very time intensive, such as the repeated checks for existing connections between two nodes, which has massively contributed to the amortized run times witnessed in Table 3.

5 Conclusion

The poorly optimized K-Means code provided by the authors was easily beaten by a basic K-Means clustering algorithm provided by Scikit Learn. While the results of the original paper were replicable, the baseline was neither the simplest, nor fastest implementation of K-Means clustering.

HDBSCAN deletion proves to be an intriguing method to research further in the future. Based on time available to spend on this project, it was difficult to write an algorithm that was fully-optimized. Some of the checks in the algorithm were very time-intensive, such as the checks for neighboring connections while reconstructing the MST. Additionally, the MRD metric used in the deletion algorithm is an approximation of the MRD calculated during training, and it would be fruitful to conceive of an efficient way to recalculate the MRD of nodes of interest after a deletion.

6 Statement of Contribution

Chris Harper: Introduction, Related Works, baseline improvement using sklearn
Suzuka Kobuku: Various parts of project report, implementation of Kmeans++, baseline improvements with Kmeans++, Q-Kmeans, and DC-Kmeans
Evan Savage: Various parts of project report, HDBSCAN deletion algorithm, survey of clustering algorithms to implement

References

[1] Antonio Ginart, Melody Y. Guan, Gregory Valiant, James Zou *Making AI Forget You: Data Deletion in Machine Learning*. Stanford University, 2019. https://arxiv.org/abs/1907.05012

[2] Campello, Ricardo J. G. B., Davoud Moulavi, and Joerg Sander. "Density-Based Clustering Based on Hierarchical Density Estimates." In Advances in Knowledge Discovery and Data Mining, edited by Jian Pei, Vincent S. Tseng, Longbing Cao, Hiroshi Motoda, and Guandong Xu, 160–72. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2013. https://doi.org/10.1007/978-3-642-37456-2_14.

[3] Ester, M., H. P. Kriegel, J. Sander, and X. Xu, "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining, Portland, OR, AAAI Press, pp. 226-231. 1996

[4] "How HDBSCAN Works — Hdbscan 0.8.1 Documentation." Accessed December 14, 2019. https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html.

[5] Chris Piech. "CS221 K-Means". Stanford University, 2013. https://stanford.edu/~cpiech/cs221/handouts/kmeans.html

[6] David Arthur, Sergei Vassilvitskii. "k-means++: The Advantages of Careful Seeding". Stanford University. http://ilpubs.stanford.edu:8090/778/1/2006-13.pdf

[7] Yinzhi Cao, Junfeng Yang. "Towards Making Systems Forget with Machine Unlearning". Columbia University, 2015. https://www.ieee-security.org/TC/SP2015/papers-archived/6949a463.pdf

[8] Chuan Guo, Tom Goldstein, Awni Hannun, Laurens van der Maaten. "Certified Data Removal from Machine Learning Models". Facebook AI, 2019. https://arxiv.org/pdf/1911.03030.pdf

[9] Lucas Bourtoule, Varun Chandrasekaran, Christopher Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, Nicolas Papernot. "Machine Unlearning". University of Toronto, University of Wisconsin, 2019. https://arxiv.org/pdf/1912.03817.pdf

[10] Fränti, P., Sieranoja, S. (2019). How much can k-means be improved by using better initialization and repeats? Pattern Recognition, 93, 95–112. doi: 10.1016/j.patcog.2019.04.014