Parameter Tunning and Model Improvement on Efficient Data Deletion in Machine Learning

Zhenghua Chen
McGill University
zhenghua.chen@mail.mcgill.ca, 260783959

Xiaobin Shang
McGill University
xiaobin.shang@mail.mcgill.ca, 260786194

Yue Lyu
McGill University
yue.lyu@mail.mcgill.ca, 260762354

Abstract

We validated the work stated in the paper "Making AI Forget You: Data Deletion in Machine Learning"[1] by reproducing their results. We found that our results roughly aligned the trends of their results but with some minor fluctuations. We further explored their proposed models by fine tuning their hyperparameters. The effects of number of iterations and tree depth on the performance of corresponding proposed models were investigated. We further optimized DC-K-means model and proposed a new competitive weighted DC-K-means model, which has better statistical performance on some datasets at a minor cost of runtime efficiency.

1. Introduction

Researchers often encounter the situation that some participants involved in past experiments requested to withdraw their data from the database. People have the legal right to decide at any time whether their data can be used or not, and researchers also need to adjust their experiments and results after erasing these data. As this situation becomes more common, developing a tool to give a flexible control to the participants over their individual data, which also doesn't exhaust the researchers, grows to be important. In some fields of study, removing a few samples from the database may be a challenge. In machine learning, if data scientists attempt to thoroughly erase a few data, they may need to retrain their model based on the huge dataset left and spend a lot of time and computation. Thus, after rigorously formulating the problem and defining the notion of efficient data deletion, Anronio, Melody, Gregory and James in Stanford University proposed two algorithms using k-means clustering and derived their running time to improve the efficiency of data deletion.[1] They also verified the methods on six different datasets and achieved over 100X speedup on average compared to k-means baseline(i.e. a k-means++ seeding followed by Lloyd's algorithm). In this project, we concisely showed the methods proposed by the above researchers and reproduced the results by running their algorithm on the same datasets. Then we investigated deeply on the essential parameters in the algorithm and tested different values numerous times to gain a relative optimization. After that, we analysed our reproduced results and the runtime of tuned models compared with the original results, and attempted to provide an improved model based on our findings.

2. Related work

Lloyd's algorithm and quantized k-means and DC-k-means: In Lloyd's paper "Least Squares Quantization in PCM"[2], he defined quantization more explicitly and used minimum average quantization noise power as an optimization criterion. Utilizing this method, the researchers in [1] proposed a quantized variant to solve the deletion efficient problem in k-means clustering, and is called quantized k-means (Q-k means) . They proved the deletion efficiency of the algorithms by showing hat it is most likely to gain constant quantized centroids with respect to deletions at each iteration. Also they proposed Divide-and -Conquer k-means algorithm (DC-k means) which works as a n-ary tree of height 1. The general idea is to divide the dataset into n small parts and solve each part as an independent k-means instance, then merged them recursively. Besides, the experiments on depth-1 DC-k means revealed a compelling trade-off between deletion time and statistical performance.

Metrics for evaluating clustering performance: In

this project we utilized the same three metrics as in the paper [1]. First, optimization loss is a commonly used tool which is the sum of square Euclidean distances from each datapoint to its nearest centroid in clustering. The second metric, Silhouette Coefficient, measures how dense each cluster is and how well-separated different clusters are. A higher score indicates a denser, more well-separated cluster and its value is between -1 and 1. Third metric is called Normalized Mutual Information. It measures the agreement of the assigned clusters to the ground-truth labels, up to permutation. Its score is less than 1 (and equals to 1 when the assignments are perfect). Higher scores indicate better agreement between clusters and ground-truth labels.

3. Dataset and setup

3.1. Description of datasets

We have five real, publicly available datasets described below, where N is number of examples in each dataset, D is the dimension of features, and K is number of classes in each dataset.

Celltype[3] (N = 12,009, D = 10, K = 4): Celltypes consists of 10 feature dimensions, 12,009 single cell RNA sequences from a mixture of 4 cell types.
Covtype[4] (N = 15,120, D = 52, K = 7): Covtype consists of 15,120 samples of 52 cartographic variables such as elevation and hillshade at various times of day for 7 forest cover types.
MNIST[5] (N = 60,000, D = 784, K = 10): MNIST consists of 60,000 images of isolated, normalized, handwritten digits. The task is to classy if each 28×28 image in to one of the ten classes.
Postures[6] (N = 74,975, D = 15, K = 5): Postures consists of 74,975 motion capture recordings of users performing 5 different hand postures with unlabeled markers attached to a left-handed glove.
Botnet[7] (N = 1,018,298, D = 115, K = 11): Botnet contains statistics summarizing the trafc between different IP addresses for a commercial IoT device (Danmini Doorbell). We aim to distinguish between benign trafc data (49548 instances) and 11 classes of malicious trafc data from botnet attacks, for a total of 1018298 instances.
Gaussian (N = 100,000, D = 25, K = 5): Gaussian consists of 5 clusters, each generated from 25-variate Gaussian distribution centered at randomly chosen locations in the unit hypercube. 20,000 samples are taken from each of the 5 clusters, for a total of 100,000 samples. Each Gaussian cluster is spherical with variance of 0.8.

3.2. Preprocessing of datasets

We preprocessed all datasets by a minmax scaling in order to map them into the unit hypercube. Since we clustered input datapoints with respect to distance, minmax scaling is important to transform each dimension of features into the same scale and reduce noise. In practice, the scaling of a dataset can change due to deletions. However, this is a minor concern as only a small number of extrenal datapoints affect the scale. Retraining from scratch when these points come up as a deletion request does not impact asymptotic runtime, and has a negligible impact on empirical runtime.

4. Proposed approach

We first validated the deletion efficient work given by [1] by reproducing their results on all their chosen datasets. We further explored the effect of hyperparameters on their model performance and came up with an optimized DC-k-means model based on their work. Due to limited computation power, when experimenting on hyperparameters we usually conducted our experiments on three chosen datasets: Celltype, Covtype, and MNIST.

To evaluate the statistical and runtime performance of proposed models, we utilized the same evaluation metrics mentioned in their paper [1], which included loss, silhouette coefficient, normalized mutual information (NMI), and amortized runtime.

4.1. Reproduce results

The results proposed [1] were reproduced by testing their proposed models following the instructions in the paper. We set hyperparameters by heuristic suggested in their paper. Tables and histograms were made to compare the reproduced and proposed results.

4.2. Effect of number of iterations on performance in proposed models

We tested a few different iteration numbers: 10, 50, 100 and kept the other part of the models unchanged and compared their statistical performance and runtime respectively on three chosen datasets.

4.3. Effect of tree depth on performance in DC-k-means model

In our referenced paper [1], they mainly focused on depth-1 tree structure in DC-k-means model. We further explored deeper tree structures in DC-k-means model on three chosen datasets, Celltype, Covtype, and MNIST. To ensure fairness, we fixed to include 64 leaf-nodes at bottom layer for each model. We built a depth-1 DC-k-means model with 1 root node and 64 leaf-nodes; a depth-2 DC-k-means model with 1, 8, 64 nodes at each layer; and a depth-3 DC-k-means model with 1, 4, 16,

64 nodes at each layer. Then, we evaluated statistical and runtime performances of our 3 models on chosen datasets.

4.4. Weighted DC-k-means model

For DC-k-means model, we further optimized their proposed structure and implemented the weighted DC-k-means model, where we weighted centroids based on cluster mass as they propagated up the tree. In our weighted DC-k-means model, we first randomly assigned N input examples with equal chance to one of leaf-nodes, as what was done before. The assigned centroids in the leaf-nodes were then propagated up to parent node as a new datapoint, which was weighted by the number of datapoints assigned to it. Thus, at root node of the tree, each centroid had a mass weight equaled to total number of datapoints assigned to it in leaf-nodes. We believed the weighted DC-k-means model would have better statistical clustering performance with negligible runtime sacrifice, since it better summerized and operated on the input data with a cluster mass weight propagated up to higher level. We also implemented a weighted k-means varient, where each datapoint was weighted by a given integer. We randomly chose k centroids for N datapoints each with probability (weight $\times$ minimum distance) to other centroids. Then, we assigned each datapoint to its nearest centroid, and updated centroids with weighted mean value of all datapoints in the same clustering.

We recapitulated our weighted DC-k-means model as Algorithm 1 below.

.

Algorithm 1: Weighted DC-k-means

Input: data matrix $D \in R^{n \times d}$;
Parameters: k clusters: $k \in N$; a list of tree width at each level: $w \in N^h$, tree height: $h \in N$;
Initialize a tree, T, of height h and w_i number of nodes at each level, such that each node has a pointer to a dataset and centroids;
for $i = 0$ **to** n **do**
    Select a leaf node uniformly at random;
    node.dataset.add(D_i);
    node.data_weights.add(1);
end
for $l = h$ **to** *0* **do**
    foreach *node in level l* **do**
        c $\leftarrow$ k-means_variate(node.dataset, k, T, node.data_weights);
        w $\leftarrow$ number of datapoints assigned to each centroid c_i;
        node.centroids $\leftarrow$ c;
        node.centroids_weights $\leftarrow$ w;
    end
    if $l > 0$ **then**
        node.parent.dataset.add(c);
        node.parent.data_weights.add(w);
    else
        save all nodes as metadata;
        return c;
    end
end

5. Experiments and Results

5.1. Reproduce results

Tables 1-4 are the results that we reproduced by following the procedures and algorithms suggested by [1]. By comparing with the proposed results in [1], as shown on fig 1-4, we found that most of reproduced results were close to the proposed results and they performed the same trends. There exists an error on Table 2 of the paper that there are 2 rows of Gaussian results. Base on the results we reproduced, we speculated that the second Gaussian row should be MNIST row. We also found that the reproduced amortized runtime was a bit different than the proposed runtime. A possible reason is that our running environment and hardware were different from the research group and caused the runtime to be different. The difference is reasonable and acceptable. As we expected, the proposed results are reproducible and the proposed algorithms are valid.

Dataset	k-means	Q-k-means	DC-k-means
Celltype	1.0±0.015	1.213±0.157	1.601±0.100
Covtype	1.0±0.031	1.022±0.022	1.006±0.027
MNIST	1.0±0.003	1.05±0.008	1.004±0.005
Postures	1.0±0.010	1.000±0.002	1.027±0.017
Botnet	1.0±0.179	1.121±0.122	1.184±0.121

Table 1: Optimization loss ratios of our proposed methods over the k-means++ baseline

Dataset	k-means	Q-k-means	DC-k-means
Celltype	0.398±0.009	0.419±0.039	0.504±0.022
Covtype	0.201±0.014	0.234±0.019	0.243±0.038
MNIST	0.064±0.007	0.063±0.009	0.069±0.005
Postures	0.095±0.006	0.104±0.004	0.096±0.012
Botnet	0.601±0.034	0.603±0.022	0.614±0.013

Table 2: Silhouette Coefficients (higher is better)

Dataset	k-means	Q-k-means	DC-k-means
Celltype	0.344±0.013	0.324±0.045	0.226±0.015
Covtype	0.325±0.012	0.334±0.021	0.326±0.006
MNIST	0.498±0.012	0.468±0.021	0.482±0.019
Postures	0.163±0.018	0.160±0.013	0.173±0.006
Botnet	0.697±0.035	0.712±0.037	0.701±0.025

Table 3: Normalized Mutual Information (higher is better)

Dataset	k-means	Q-k-means	DC-k-means
Celltype	2.655±0.196	0.099±0.071	0.179±0.012
Covtype	3.416±0.131	0.457±0.338	0.301±0.027
MNIST	34.26±2.754	19.47±2.843	1.988±0.076
Postures	17.31±1.311	3.323±1.59	1.021±0.061
Botnet	328.5±39.96	33.13±21.63	26.251±1.55

Table 4: Amortized Runtime in Online Deletion Benchmark (Train once + 50 Deletions)

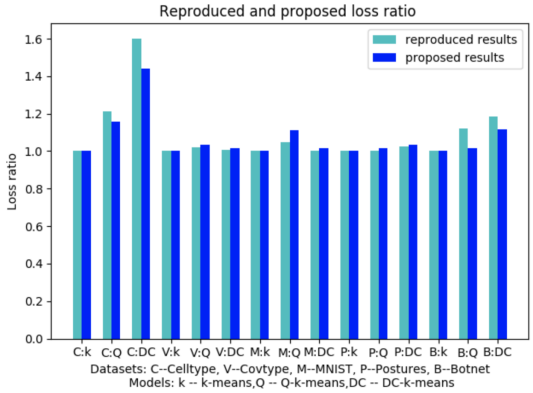

Figure 1: Comparison between the reproduced and the proposed loss ratio of the three models on the five datasets

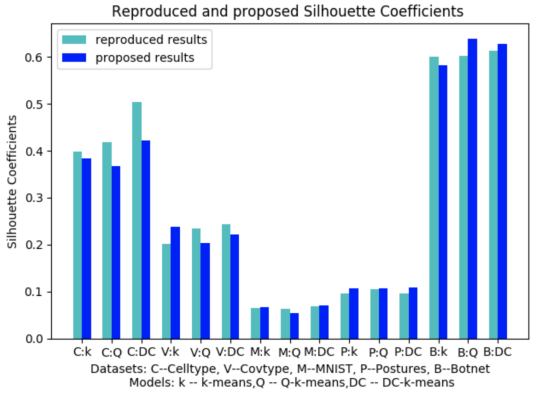

Figure 2: Comparison between the reproduced and the proposed Silhouette Coefficients of the three models on the five datasets

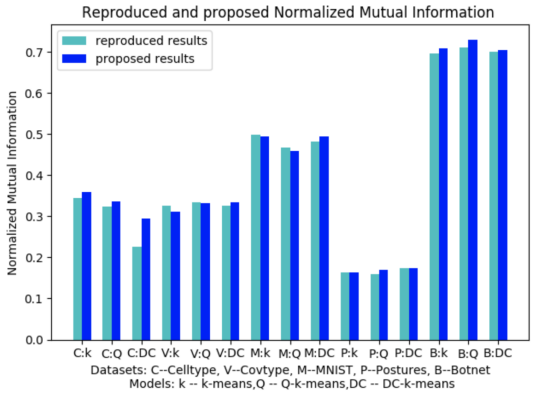

Figure 3: Comparison between the reproduced and the proposed Normalized Mutual Information of the three models on the five datasets

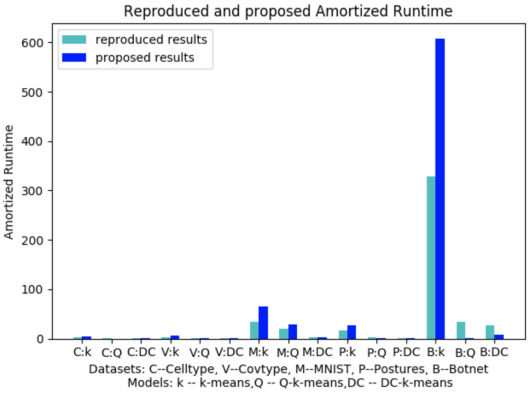

Figure 4: Comparison between the reproduced and the proposed Amortized Runtime of the three models on the five datasets

5.2. Effect of number of iterations on performance in proposed models

As described in proposed approach, we summerized our results in the following figures.

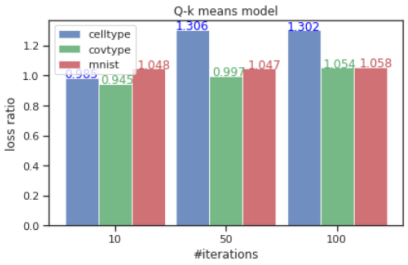

Figure 5: Q-k-means loss ratios (to k-means) at different iteration times on celltype, covtype and MNIST datasets.

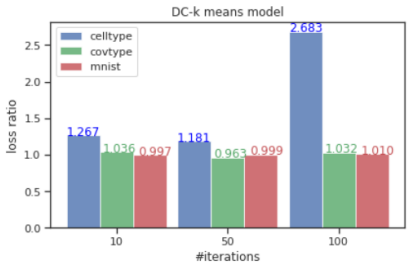

Figure 6: DC-k-means loss ratios (to k-means) at different iteration times on celltype, covtype and MNIST datasets

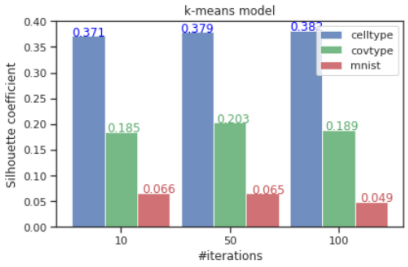

Figure 7: Silhouette coefficient values of k-means at different iteration times on celltype, covtype and MNIST datasets.

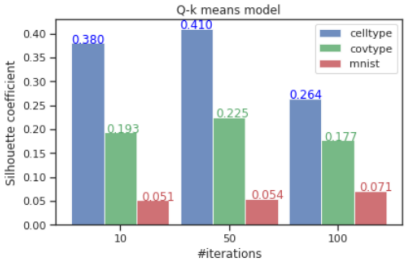

Figure 8: Silhouette coefficient values of Q-k-means at different iteration times on celltype, covtype and MNIST datasets.

.

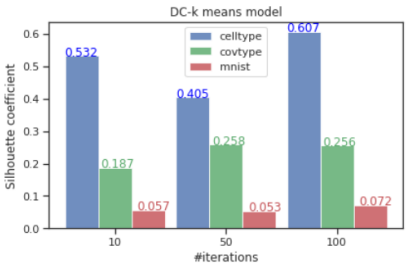

Figure 9: Silhouette coefficient values of DC-k-means at different iteration times on celltype, covtype and MNIST datasets.

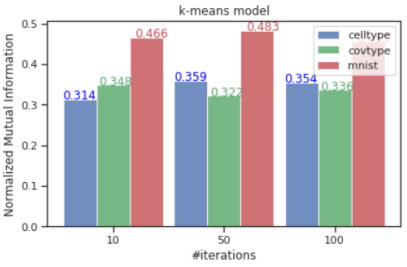

Figure 10: Nomalized mutual information of k-means at different iteration times on celltype, covtype and MNIST datasets.

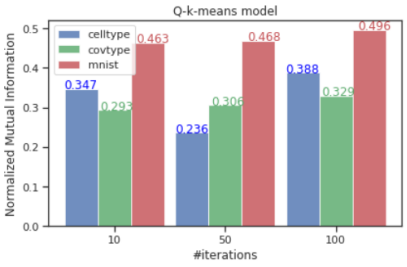

Figure 11: Nomalized mutual information of Q-k-means at different iteration times on celltype, covtype and MNIST datasets.

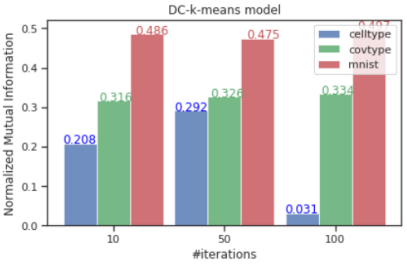

Figure 12: Nomalized mutual information of DC-k-means at different iteration times on celltype, covtype and MNIST datasets

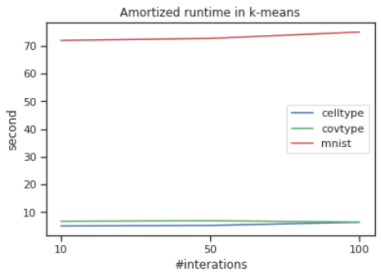

Figure 13: Amortized runtime of k-means at different iteration times on celltype, covtype and MNIST datasets.

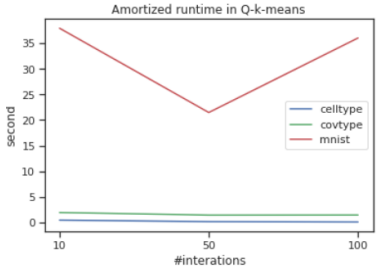

Figure 14: Amortized runtime of Q-k-means at different iteration times on celltype, covtype and MNIST datasets.

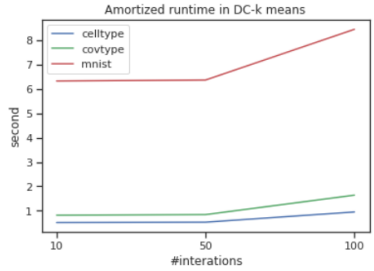

Figure 15: Amortized runtime of DC-k-means at different iteration times on celltype, covtype and MNIST datasets.

When the number of iterations increases, we see that in general, the loss ratios of Q-k-means and DC-k-means model over k-means model have increased. For covtype and MNIST dataset, it increases slightly but for celltype dataset the loss ratio increases significantly. The silouette coefficients of k-means model and DC-k-means model generally increases as the number of iterations goes up. This phenomenon is more obvious in celltype and covtype dataset, and in DC-k-means model the coefficient value reaches up to 0.607 for the celltype dataset. Q-k-means model shows a general decrease in silhouette coefficient value on celltype and covtype datasets, but has a small increase on MNIST dataset. The normalized mutual information of the three models all increases for the celltype dataset when number of iterations grows from 10 to 100. However, in DC-k-means model, the information of celltype dataset surprisingly decreases to 0.031. When analyzing the amortized runtime of the models, we noticed that the runtime fluctuates differently among different datasets and models and the runtime for celltype is much longer than the other datasets. Overall, DC-k-means has the shortest amortized runtime compared with the other two models at different iterations. Besides, K-means and DC-k-means model both indicate more obvious increases in runtime compared with Q-k-means over different iteration times.

5.3. Effect of tree depth on performance in DC-k-means model

As discussed in proposed approach, we experimented on DC-k-means model with depth-1, depth-2, and depth-3 tree structures on three chosen datasets, Celltyp, Covtype, and MNIST. We summerized our results in the following 3 figures for 3 datasets.

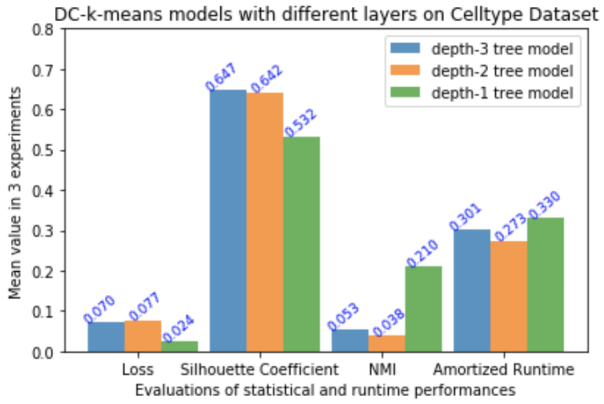

Figure 16: Comparison between performance metrics of Weighted DC-k-means model and DC-k-means model on MNIST dataset.

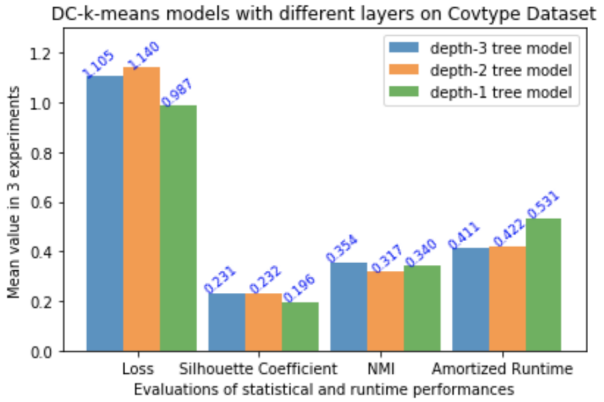

Figure 17: Comparison between performance metrics of Weighted DC-k-means model and DC-k-means model on MNIST dataset.

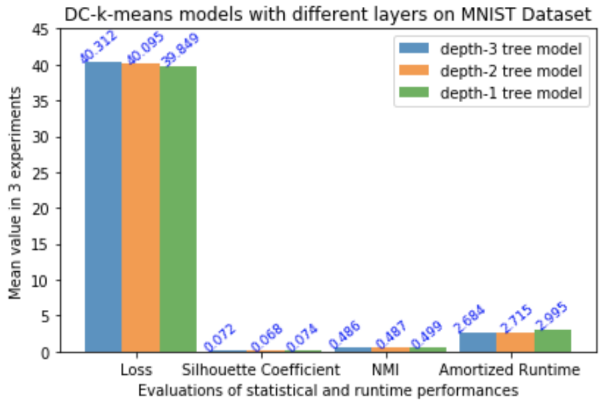

Figure 18: Comparison between performance metrics of Weighted DC-k-means model and DC-k-means model on MNIST dataset.

We can see that in dataset Covtype, DC-k-means model with deeper tree structure will generate better statistical and runtime performance. It's also important to notice that deeper tree structure in DC-k-means model in general reduce amortized runtime of training model and deleting datapoints from trained model, since we will recluster fewer datapoints at higher level when we merge our modified branch with other branches.

5.4. Weighted DC-k-means model

We propose to compare the statistical and runtime performance of Wieghted DC-k-means model and original DC-k-means model, where we evaluate our models on loss, silhouette coefficient by 10000 random sampled datapoints, normalized mutual information score, and amortized runtime of 20 deletions. To control variates, we only explore depth-1 trees with w leaves, where w is chosen by heuristic as did in our referenced paper [1]. We build the same depth-1 tree with w leaves structure for two models. Then, we run 3 experiments of two models (Weighted DC-k-means model and DC-k-means model) on each chosen datasets (Celltype, Covtype, Postures, and MNIST) and compare their performance on proposed evaluation metircs. The following 4 figures shows the comparison of performance of 2 models on 4 chosen datasets.

As shown in the results, we can see that Weighted DC-k-means model is able to generate better statistical clustering performance on some datasets at a cost of runtime efficiency. We find that in relative large dimension datasets (like Postures and MNIST), Weighted DC-k-means model has better performance than original DC-k-means model. In other 2 small datasets, Weighted DC-k-means model doesn't have obvious advantage. We can treat Weighted DC-k-means model as a another competitive approach that is bounded by the same asymptotic runtime and slightly higher practical runtime as DC-k-means model, where we can perform model selection on specific dataset to find the relative best model.

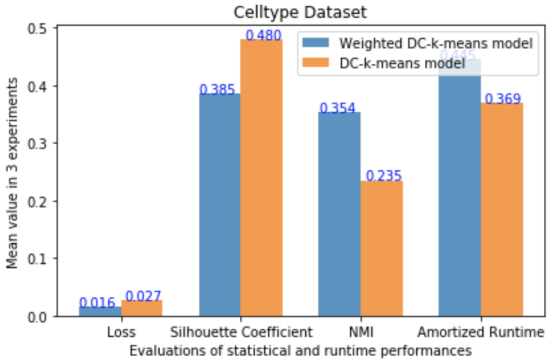

Figure 19: Comparison between performance metrics of Weighted DC-k-means model and DC-k-means model on Celltype dataset.

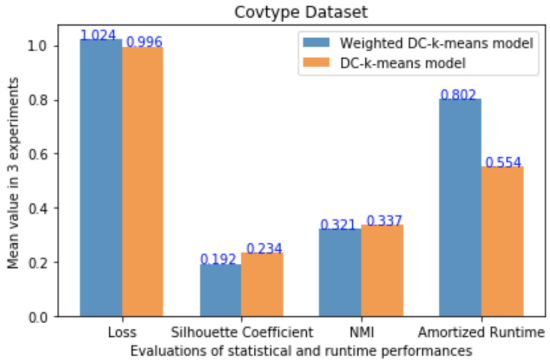

Figure 20: Comparison between performance metrics of Weighted DC-k-means model and DC-k-means model on Covtype dataset.

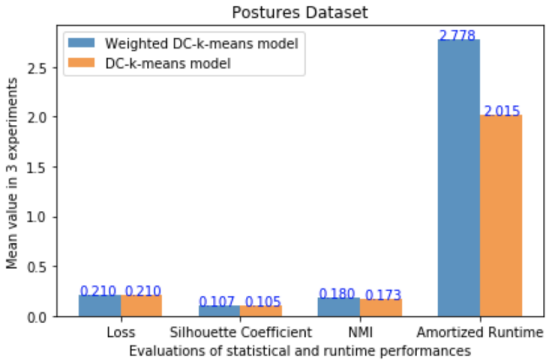

Figure 21: Comparison between performance metrics of Weighted DC-k-means model and DC-k-means model on Postures dataset.

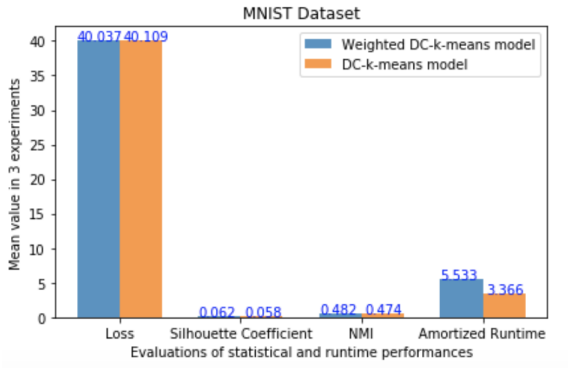

Figure 22: Comparison between performance metrics of Weighted DC-k-means model and DC-k-means model on MNIST dataset.

6. Discussion and Conclusion

6.1. Discussion:

Due to limited computation power, we did our experiments only on a chosen subsets of acquired datasets. For some evaluation parameters (like silhouette coefficient), we calculate them only based on a random sample with fixed size from all datapoins, which is prone to induce noise and fluctuations. In the future, we should conduct more thorough experiments on more datasets to derive conclusions with better precision and genralization ability.

6.2. Conclusion:

Through a large mount of attempts, we found that under different values of parameters, the statistical and runtime performances of Q-k-means and DC-k-means vary with different datasets. It's hard to find general optimal parameter values which make the models perform well on all distinct datasets. However, we did discover that DC-k-means model outperforms the other two based on silouette coefficient, normalized mutual information and amortized runtime. Among most datasets and different parameters, DC-k-means clusters the data well compared with k-means and Q-k-means

7. Statement of Contributions

Yue Lyu: Implement DC-k-means model and test its performance. Explore effect of tree structures on performance of DC-k-means model. Compose final report.
Zhenghua Chen: Explore effect of number of iterations on performance of proposed models. Compose final report.
Xiaobin Shang: Explore effect of epsilon on performance of Q-k-means model. Compose final report.

References

[1] *"Making AI Forget You: Data Deletion in Machine Learning" by Antonio A. Ginart, Melody Y. Guan, Gregory Valiant, and James Zou.*

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

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

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

[5] *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.*

[6] *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.*

[7] *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.*