Sampling to Speed Up Clustering Algorithms

Likhita Baswani
likhita.b@iitgn.ac.in

Paras Jain
jain.paras@iitgn.ac.in

Shridhar Pawar
sominath.ps@iitgn.ac.in

Abstract

With an enormous increase in the size of datasets in recent times, many models that have a superlinear algorithmic complexity in the input size have become computationally challenging in the presence of millions of data points. Coresets are compact subsets of these massive datasets on which the computational models can be trained to achieve considerably similar results as compared to those that are obtained by training the same models on the entire datasets. [5] As clustering is one such complex algorithm, coresets can be used to scale up clustering models to larger datasets. We provide algorithms to construct lightweight coresets, uniform coresets, and adaptive sampling for k-means++ clustering. With this, we study the relative error in the results when the k-means++ clustering is trained on each of these coresets versus the FULL dataset. Further, we report the relative speedup compared to FULL. Additionally, we compare the above three sampling methods.

Keywords: coresets, k-means++ clustering, lightweight, uniform, adaptive

1 INTRODUCTION

From speech-recognition and product recommendation to traffic prediction and self driving cars, Machine Learning is ruling the world. These developments have been periodically enabled by larger and more complex datasets. However, with the increase in the size of data sets, many traditional algorithms fail to scale to such massive data sets. For instance, the algorithms with superlinear time complexity become computationally infeasible with millions to billions of input data points. In addition, many algorithms require accessing the data from a single machine multiple number of times. This not only increases the algorithmic complexity when dealing with larger datasets, but the whole dataset may also get disturbed across a cluster of machines that is also accessing it.

Over the years, many methods have been proposed to compute using big data. One of the simplest and primary methods would be to rely on advanced hardware and infrastructure. However, this solution gets ruled out in most cases due to unaffordability. Other options include reducing the computational complexity by manipulating the input size of the data sets. This idea brings us to finding relevant subsets(coresets) of the original dataset, on which the computations can be carried out. In the recent years, coresets have been successfully created for many clustering algorithms. In this project, we will focus on coresets for k-means++ clustering algorithm as follows:

- Create a first type of coresets - Lightweight coresets by Bachem et al. [5]. It defines a lightweight form of coresets whose samples can be considered to provide a good fit with entire data set.
- The second type of coresets construction is Adaptive Sampling by Feldman et al. [2].
- Apart from these, we also use Uniform sampling as a comparison baseline for the performance of all the coresets.

2 BACKGROUND

k-Means++ Clustering

k-means++ Clustering is one of the Machine Learning algorithms that makes inferences from the input data points without referring to any known outcomes [4]. The objective is to come up with a fixed number(k) of clusters such that similar data points are grouped together to discover underlying patterns. To achieve this:

- The algorithm starts with a random centroid among the k centroids
- A data point whose distance is greatest from the existing centroid is chosen as the new centroid. This is repeated until the first set of k centroids are chosen
- Each data point is then allocated to its nearest centroid, thereby forming k clusters
- A new set of centroids is obtained by calculating the 'means' of all data points in each cluster
- The above two steps are repeated until either the desired number of iterations is met or there is no significant change in the centroid locations

Mathematically, if χ is a set of points in $\mathbb{R}$, the clustering problem is to find a set Q of k cluster centers in $\mathbb{R}$ such that the quantization error ϕ_χ(Q) is minimized, where

$$\phi_\chi(\mathrm{Q}) = \sum_{x\in\chi} d(x,Q)^2 = \sum_{x\in\chi} min_{q\in Q}||x-q||_2^2$$

Coresets for k-Means++

An increase in the data set size(χ) and the number of clusters(k) leads to increased number of iterations of the k-means++ clustering algorithm. In this scenario, execution time remains a huge obstacle. Therefore, the algorithm requires a scalable solution to manage large data sets. For this, instead of solving on the entire χ, clustering problem can be applied to a subset of χ. However, finding the relevant subset to compute upon is a challenge. Thus, the goal is to construct a (k, ϵ)coreset that can be defined as a weighted set C given any Q $\subset \mathbb{R}$ with |Q| $\leq$ k such that:

$$|\phi_\chi(\mathrm{Q}) \text{ - } \phi_C(\mathrm{Q})| \leq \epsilon\phi_\chi(Q)$$

which is equivalent to

$$(1-\epsilon)\phi_\chi(\mathrm{Q}) \leq \phi_C(\mathrm{Q}) \leq (1+\epsilon)\phi_\chi(\mathrm{Q})$$

This gives us a (1+ϵ) approximation subset, for some $0 < \epsilon < 1$.

3 CORESET CONTRUCTION METHODS

In recent years, many coreset creation techniques for clustering problems have been suggested. As a result, several approaches such as exponential grids, bounding points, and dimension reduction have been examined. In this project, we will concentrate on an approach that has recently been employed in research: sampling-based methodologies.
Finding coresets based on sampling is a straightforward and quick technique. This is the process of selecting a subset of people from a population or original set based on a probability or distribution. It does, however, necessitate a large number of mathematical proofs and

theorems. There are several studies on sampling-based coreset constructions, just as there are for other approaches. In this study, we use two of the most successful and well-proven methods: Adaptive Sampling and the Lightweight Coreset, in addition to naive or uniform sampling as the baseline for comparisons.

3.1 Adaptive Sampling

The core idea is to create a sample set(C) that is a rough solution that can be used to bias random sampling. An iterative algorithm samples a limited number of points and removes half of the data set χ nearest to the sampled points in the initial phase. Later, the sample is skewed by using probabilities that are roughly proportional to the squared distance between each point in χ and C.

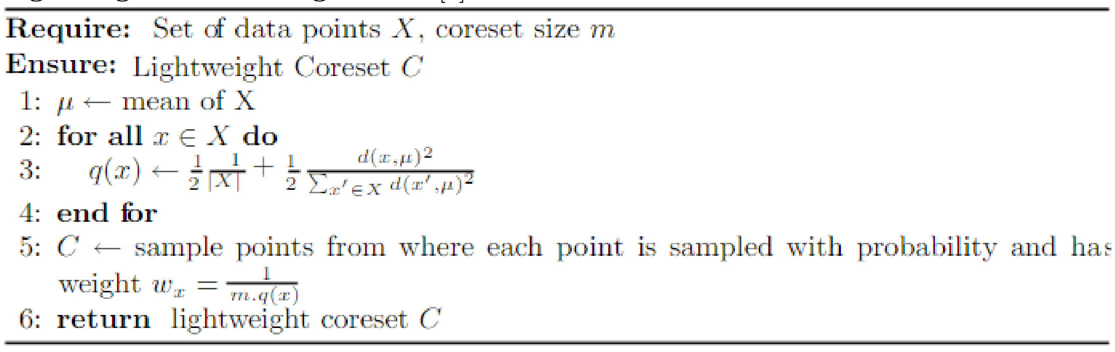
Adaptive Sampling Algorithm: [3]

```
Require: Data set $D, \varepsilon, \delta, k$
Ensure: Coreset $C = \{(\gamma(x_1), x_1), ..., (\gamma(x_{|C|}), x_{|C|})\}$
 1: $D' \leftarrow D; B \leftarrow \emptyset$;
 2: while $(|D'| > 10dk \ln(1/\delta))$ do
 3:     Sample set $S$ of $\beta = 10dk \ln(1/\delta)$ points uniformly at random from $D'$;
 4:     Remove $\lceil |D'|/2 \rceil$ points $x \in D'$ closest to $S$ (i.e. minimizing $dist(x, S)$) from $D'$;
 5:     Set $B \leftarrow B \cup S$;
 6: end while
 7: $B \leftarrow B \cup D'$;
 8: for each $b \in B$ do
 9:     $D_b \leftarrow$ the points in $D$ whose closest point in $B$ is $b$. Ties broken arbitrarily;
10: end for
11: for each $b \in B$ and $x \in D_b$ do
12:     $m(x) \leftarrow \lceil \frac{5}{|D_b|} + \frac{dist(x,B)^2}{\sum_{x' \in D} dist(x',B)^2} \rceil$;
13: end for
14: Pick a non-uniform random sample $C$ of $10\lceil dk|B|^2 \ln(1/\delta)/\varepsilon^2 \rceil$ points from $D$,
    where for every $x' \in C$ and $x \in D$, we have $x' = x$ with probability $\frac{m(x)}{\sum_{x' \in D} m(x')}$;
15: for each $x' \in C$ do
16:     $\gamma(x') \leftarrow \frac{\sum_{x \in D} m(x)}{|C|.m(x')}$;
17: end for
18: return $C$
```

3.2 Lightweight Coresets

'Importance sampling' is used in the creation of the Lightweight coresets. The mean of the data is calculated and then used to construct the importance sampling distribution *q(x)*. Finally, m points from X are sampled with probability *q(x)* and the weight *1/m*q(x)* is allocated to points. The algorithm only goes through the data set twice, resulting in a total computational complexity of *O(nd)*. In the case where k is relatively even higher, there is no further linear dependence on the number of clusters k.

Lightweight Coreset Algorithm: [3]



```
Require: Set of data points $X$, coreset size $m$
Ensure: Lightweight Coreset $C$
1: $\mu \leftarrow$ mean of X
2: for all $x \in X$ do
3:     $q(x) \leftarrow \frac{1}{2}\frac{1}{|X|} + \frac{1}{2}\frac{d(x,\mu)^2}{\sum_{x' \in X} d(x',\mu)^2}$
4: end for
5: $C \leftarrow$ sample points from where each point is sampled with probability and has
   weight $w_x = \frac{1}{m.q(x)}$
6: return lightweight coreset $C$
```

4 DATA SETS

We consider the k-means++ clustering problem on three different data sets for both k = 100 and k = 200:

1. heart.csv:
   Heart Disease Indicators - 319,795 samples with 18 features indicating if a person has a heart disease given his/her physical and mental health indicators, habits, and personal traits. [6]
2. product.csv:
   Product Classification - 238,170 products belonging to 12 different categories supplied by 652 electronic stores [1]
3. credit.csv:
   Credit Card Customers - 18 features of 8950 customers information used for identifying loyal customers, customer segmentation, and targeted marketing [7]

5 METHODOLOGY

The data is first pre-processed and cleaned before applying k-means++ to it. Cleaning includes deleting null values, removing outliers, and changing data types. In pre-processing of data, each column's instances are standardized by using standard scaling. Object type columns are changed to float using label encoding. The experiment on each data set includes the following steps:

1. Use kmeans++ to cluster the full dataset
2. Generate samples using *Adaptive sampling* with sizes m = 1000, 2000, 3000, 4000, and 5000
3. Generate samples using *Lightweight sampling* with sizes m = 1000, 2000, 3000, 4000, and 5000.
4. Generate samples using *Uniform sampling* with sizes m = 1000, 2000, 3000, 4000, and 5000.
5. Use kmeans++ to solve the clustering problem on each sample.
6. Measure the elapsed time and cost of clustering for each sample as well as the clustering of the full dataset.
7. Finally, compute the relative error for each method and sample size with respect to the full dataset.

We executed the Uniform sampling and Lightweight coreset tests 25 times with different random seeds and computed sample averages since they are randomized. All of the tests were written in Python and executed on an Intel Core i5 machine with 3.2 GHz processors and 8 GB of memory.

6 RESULTS AND DISCUSSIONS

In this project we compare three coreset construction methods:

1. Uniform sampling, denoted by "Uniform"
2. Adaptive sampling, denoted by "Adaptive"
3. Lightweight sampling, denoted by "Lightweight"

We use relative error as the measurement for the correctness and the run-time comparison. The results are expressed as follows:

- Figures 1 and 2 plot the relation between relative error and subsample size for different datasets.
- Table 1 shows the relative error for different Coreset construction under different conditions. Here, a smaller value indicates better performance.
- Table 2 shows the time comparison between Coreset constructions. Clearly, a smaller value of time means faster.

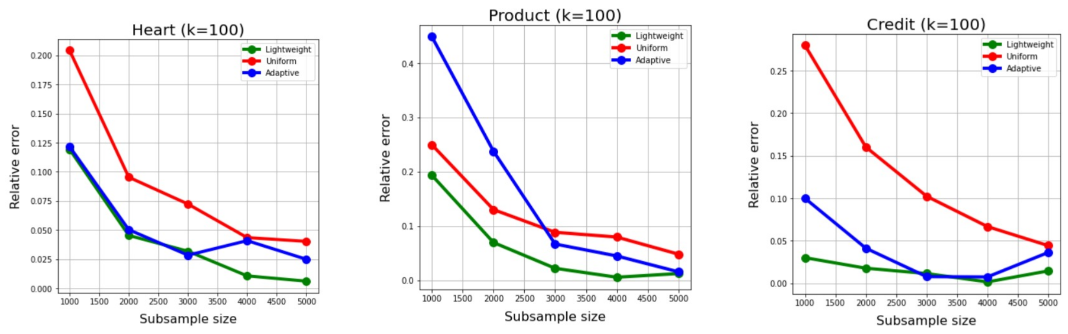


Figure 1 Relative error in relation to subsample size for Uniform, Lightweight and Adaptive Sampling for kmeans++ with k=100

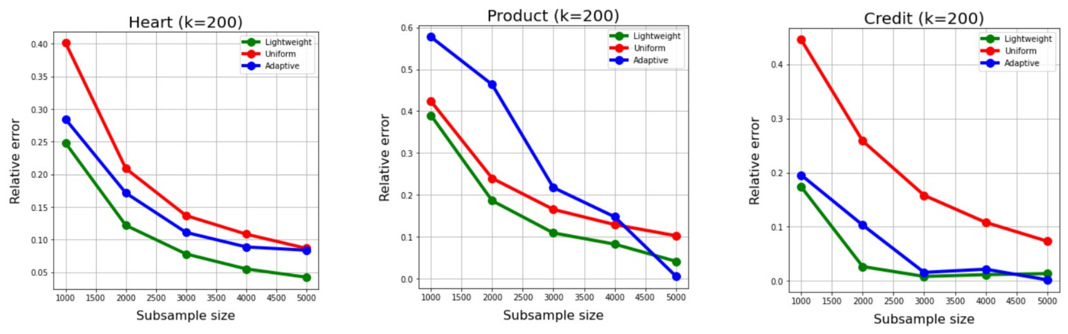


Figure 2 Relative error in relation to subsample size for Uniform, Lightweight and Adaptive Sampling for kmeans++ with k=200

Table 1 The relative error for different Coreset constructions under different conditions

| k |	Data	Method	m=1000	m=2000	m=3000	m=4000	m=5000
100	Heart	LightWeight	0.1192	0.0453	0.0318	0.0106	0.0061
		Uniform	0.2041	0.0953	0.0723	0.0434	0.0402
		Adaptive	0.1215	0.0502	0.0284	0.0408	0.0248
	Product	LightWeight	0.1932	0.0694	0.0219	0.0054	0.0121
		Uniform	0.2461	0.1399	0.1057	0.0787	0.0561
		Adaptive	0.4431	0.1730	0.0171	0.2417	0.2981
	Credit	LightWeight	0.0228	0.0177	0.0098	0.0039	0.0148
		Uniform	0.2797	0.1602	0.1023	0.0666	0.0441
		Adaptive	0.0998	0.0411	0.0076	0.0073	0.03612
200	Heart	LightWeight	0.2478	0.1218	0.0780	0.0550	0.0424
		Uniform	0.4010	0.2088	0.1368	0.1083	0.0866
		Adaptive	0.2843	0.1714	0.1110	0.0887	0.0836
	Product	LightWeight	0.3901	0.1860	0.1090	0.0822	0.0414
		Uniform	0.4295	0.2497	0.1702	0.1247	0.1091
		Adaptive	0.7306	0.2519	0.1676	0.1867	0.0345
	Credit	LightWeight	0.1706	0.0250	0.0044	0.0163	0.0116
		Uniform	0.4452	0.2584	0.1581	0.1082	0.0735
		Adaptive	0.1955	0.1032	0.0158	0.0217	0.0021

Table 2 The time comparison between Coreset constructions under different conditions

| k |	Data	Method	m=1000	m=2000	m=3000	m=4000	m=5000
100	Heart	LightWeight	0.135	0.141	0.164	0.178	0.188
		Uniform	0.074	0.091	0.111	0.127	0.148
		Adaptive	21.113	23.995	22.729	21.723	22.739
	Product	LightWeight	0.104	0.115	0.121	0.131	0.154
		Uniform	0.080	0.090	0.102	0.115	0.128
		Adaptive	51.208	48.254	49.430	58.124	55.793
	Credit	LightWeight	0.084	0.096	0.136	0.163	0.196
		Uniform	0.078	0.097	0.135	0.144	0.162
		Adaptive	15.958	15.162	16.708	16.529	17.517
200	Heart	LightWeight	0.162	0.196	0.235	0.259	0.296
		Uniform	0.119	0.151	0.189	0.224	0.238
		Adaptive	13.909	9.049	9.004	9.353	9.408
	Product	LightWeight	0.146	0.161	0.170	0.196	0.224
		Uniform	0.120	0.142	0.173	0.190	0.225
		Adaptive	48.588	49.861	53.226	51.566	49.239
	Credit	LightWeight	0.146	0.178	0.234	0.274	0.344
		Uniform	0.144	0.179	0.236	0.268	0.313
		Adaptive	14.344	15.807	14.955	13.228	17.907

6.1 Discussions

- Uniform Sampling produces large error values in the majority of scenarios, implying that Uniform Sampling is the worst coreset construction. This is understandable, given that uniform sampling is the most basic and inexperienced method. This is, however, the quickest approach.
- The relative errors for all approaches diminish as the sample size is raised, as seen in all Figs. 1, 2, and 3. If we get more points, we will be able to receive more precise coresets.
- In most circumstances, Lightweight Coreset not only performs well, but it is also quite quick; in fact, it is only slightly slower than Uniform Sampling and far faster than other approaches. Light-weight coreset, on the other hand, produces a sample with low error.
- In most circumstances, adaptive sampling produces coresets with low error, especially if the clusters are well separated.
- In some circumstances, the three sampling-based approaches (uniform, adaptive, and lightweight coreset) produce coresets with extremely large errors, whereas in other cases, they produce coresets with extremely low errors. The average errors of sampling-based approaches, on the other hand, appear to be good enough to utilize in practice.

7 CONCLUSION

We describe and analyze the following state-of-the-art coreset constructions - Adaptive Sampling and Lightweight Coreset, in this study. We compare these methods using relative errors, with uniform sampling as the baseline. All experiments are completed at a glance with sampling-based class constructions. However, the correctness of the created sample remains a major issue. Because this is a sampling-based strategy, we must ensure that the outcome is satisfactory. Finally, each strategy discussed in this study has its own set of benefits and drawbacks. Before using any of these algorithms in practice, the options 'Slow but more accurate' and 'Fast but less accurate' will be weighed.

References

[1] Leonidas Akritidis. *Skroutz dataset for evaluating product classification and categorization algorithms.* URL: `https://www.kaggle.com/datasets/lakritidis/product-classification-and-categorization?select=skroutz_aggregate.csv`. (published: 09.03.2020).

[2] Dan Feldman, Matthew Faulkner and Andreas Krause. "Scalable Training of Mixture Models via Coresets". In: *Advances in Neural Information Processing Systems*. Vol. 24. Curran Associates, Inc., 2011. URL: `https://proceedings.neurips.cc/paper/2011/file/2b6d65b9a9445c4271ab9076ead5605a-Paper.pdf`.

[3] Nguyen Hoang, Le Hong Trang and Tran Dang. "A Comparative Study of the Some Methods Used in Constructing Coresets for Clustering Large Datasets". In: *SN Computer Science* 1 (2020). DOI: `10.1007/s42979-020-00227-7`.

[4] Satyam Kumar. *Understanding K-Means, K-Means++ and, K-Medoids Clustering Algorithms.* URL: `https://towardsdatascience.com/understanding-k-means-k-means-and-k-medoids-clustering-algorithms-ad9c9fbf47ca`. (published: 11.06.2020).

[5] Bachem Olivier, Lucic Mario and Krause Andreas. "Scalable k -Means Clustering via Lightweight Coresets". In: *KDD '18, August 19–23, 2018, London, United Kingdom.* 2018, pp. 1119–1127. DOI: `10.1145/3219819.3219973`.

[6] Sakshi. *Heart Disease Indicators:ML Algorithms Comparison.* URL: `https://www.kaggle.com/code/gladerunner/heart-disease-indicators-ml-algorithms-comparison/data`. (published: 10.04.2022).

[7] Arya Shah. *A Custom Dataset For Customer Segmentation Using Clustering Techniques.* URL: `https://www.kaggle.com/datasets/aryashah2k/credit-card-customer-data`. (published: 15.05.2021).