Sampling to Speed Up Clustering Algorithms

Likhita Baswani - Paras Jain - Shridhar Pawar

19110091 19110096 19110101

Table of contents

Problem Statement

Massive data sets + Superlinearly complex algorithms = Computational infeasibility

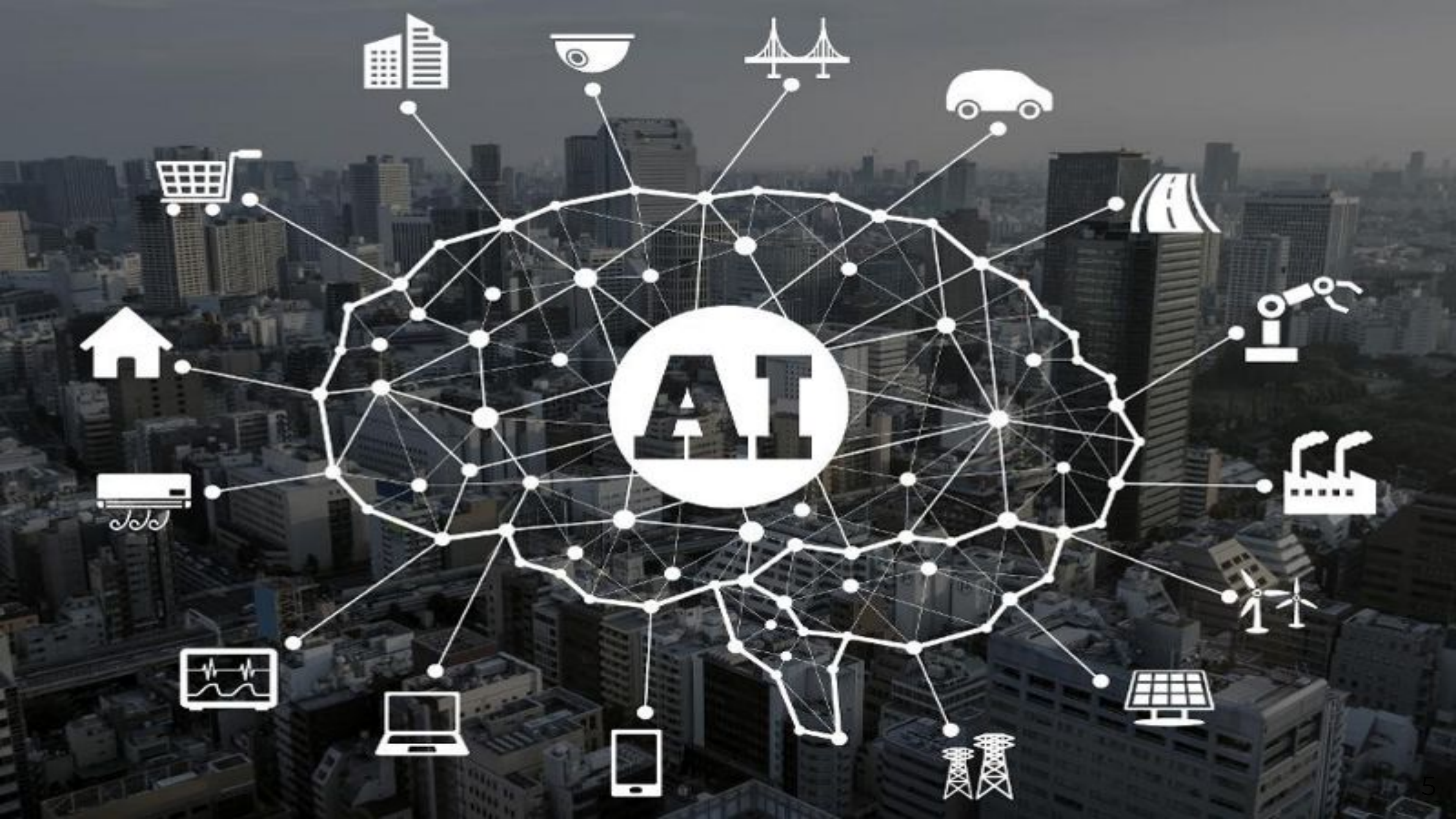
Imagine having to solve clustering problem for a large data set.

Instead of solving the problem for the entire data set, what if we could downsample the data set and get equivalent clustering results on this data subset?

These smaller samples = **Coresets**

AIM: Implement lightweight, adaptive, uniform coresets and compare their performances

02
Introduction


AI

The Problem with Big Data?

- Traditional algorithms fail to scale to such massive data sets
- Superlinear algorithms become computationally infeasible
- Accessing the data from a single machine multiple number of times
    - poor resource management + increasing time complexity
- Gets worse when the data set is accessed by a cluster of machines

Possible Solutions

- Immediate solution?
    - Rely on advanced hardware and infrastructure - EXPENSIVE!
- Input size manipulation
    - Considering relevant subsets**(CORESETS)** which give equivalent results as that of entire data set

CORESETS

Coresets are compact subsets of 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.

That is, **AlgorithmResults(Coresets) ≈ AlgorithmResults(Full Data)**

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
- The second type of coresets construction is Adaptive Sampling
- Apart from these, we also use Uniform sampling as a comparison baseline for the performance of all the coresets.

03

Coreset Construction

In recent years, many coreset creation techniques for clustering problems have been suggested.

In this project, we will concentrate on sampling-based methodologies:

This is the process of selecting a subset of people from a population or original set based on a probability or distribution. We implement the following sampling-based coreset constructions:

- Adaptive sampling
- Lightweight sampling
- Uniform sampling

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:
  - Initial Phase:
    Samples a limited number of points and removes half of the dataset χ nearest to the sampled points.
  - Second Phase:
    The sample is skewed by using probabilities that are roughly proportional to the squared distance between each point in χ and C

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.

04

Data Sets

Data for Experiment

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.
2. product.csv
3. credit.csv

	HeartDisease	BMI	Smoking	AlcoholDrinking	Stroke	PhysicalHealth	MentalHealth	DiffWalking	Sex	AgeCategory	Race	Diabetic	PhysicalActivity	GenHealth	SleepTime	Asthma	KidneyDisease	SkinCancer
0	No	16.60	Yes	No	No	3.0	30.0	No	Female	55-59	White	Yes	Yes	Very good	5.0	Yes	No	Yes
1	No	20.34	No	No	Yes	0.0	0.0	No	Female	80 or older	White	No	Yes	Very good	7.0	No	No	No
2	No	26.58	Yes	No	No	20.0	30.0	No	Male	65-69	White	Yes	Yes	Fair	8.0	Yes	No	No
3	No	24.21	No	No	No	0.0	0.0	No	Female	75-79	White	No	No	Good	6.0	No	No	Yes
4	No	23.71	No	No	No	28.0	0.0	Yes	Female	40-44	White	No	Yes	Very good	8.0	No	No	No

Figure: heart.head()

Data for Experiment

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

1. heart.csv
2. **product.csv:**
   Product Classification - 238,170 products belonging to 12 different categories supplied by 652 electronic stores.
3. credit.csv

	ProductID	ProductTitle	VendorID	ClusterID	ClusterLabel	CategoryID	CategoryLabel
0	1	amd ryzen 5 1600 box epexergastis me wraith sp...	1030	1	AMD Ryzen 5 1600 Box	696	CPUs
1	2	amd ryzen 5 1600	3964	1	AMD Ryzen 5 1600 Box	696	CPUs
2	3	amd ryzen 5 1600 box pliromi ke se eos 36 dosis	4814	1	AMD Ryzen 5 1600 Box	696	CPUs
3	4	amd ryzen 5 1600 yd1600bbaebox	4835	1	AMD Ryzen 5 1600 Box	696	CPUs
4	5	amd ryzen 5 1600 box yd1600bbaebox	2976	1	AMD Ryzen 5 1600 Box	696	CPUs

Figure: product.head()

Data for Experiment

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

1. heart.csv:
2. product.csv:
3. **credit.csv:**
   Credit Card Customers - 18 features of 8950 customers information used for identifying loyal customers, customer segmentation, and targeted marketing

	CUST_ID	BALANCE	BALANCE_FREQUENCY	PURCHASES	ONEOFF_PURCHASES	INSTALLMENTS_PURCHASES	CASH_ADVANCE	PURCHASES_FREQUENCY	ONEOFF_PURCHASES_FREQUENCY	PURCHASES_INSTALLMENTS_FREQUENCY	CASH_ADVANCE_FREQUENCY	CASH_ADVANCE_TRX	PURCHASES_TRX	CREDIT_LIMIT	PAYMENTS	MINIMUM_PAYMENTS	PRC_FULL_PAYMENT	TENURE
0	C10001	40.900749	0.818182	95.40	0.00	95.4	0.000000	0.166667	0.000000	0.083333	0.000000	0	2	1000.0	201.802084	139.509787	0.000000	12
1	C10002	3202.467416	0.909091	0.00	0.00	0.0	6442.945483	0.000000	0.000000	0.000000	0.250000	4	0	7000.0	4103.032597	1072.340217	0.222222	12
2	C10003	2495.148862	1.000000	773.17	773.17	0.0	0.000000	1.000000	1.000000	0.000000	0.000000	0	12	7500.0	622.066742	627.284787	0.000000	12
3	C10004	1666.670542	0.636364	1499.00	1499.00	0.0	205.788017	0.083333	0.083333	0.000000	0.083333	1	1	7500.0	0.000000	NaN	0.000000	12
4	C10005	817.714335	1.000000	16.00	16.00	0.0	0.000000	0.083333	0.083333	0.000000	0.000000	0	1	1200.0	678.334763	244.791237	0.000000	12

Figure: credit.head()

05

Methodology

Pre-processing

The data is first pre-processed and cleaned before applying k-means++ to it.

Pre-processing includes:

- Deleting null values
- Removing outliers
- Checking unique values.
- Standardizing each column’s instances using standard scaling.
- Changing object type columns to float using label encoding

Experiment

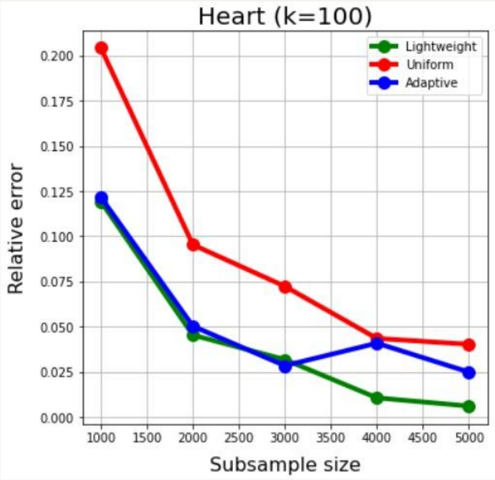
The experiment on each data set includes the following steps:

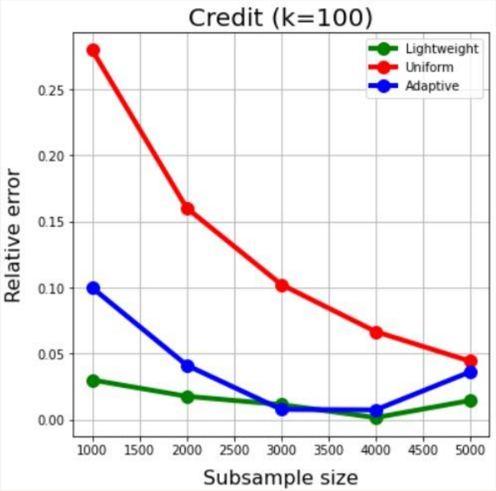
1. Use kmeans++ to cluster the full dataset
2. Generate samples of sizes m = 1000, 2000, 3000, 4000, and 5000 using:
   a. Adaptive Sampling
   b. Lightweight Sampling
   c. Uniform Sampling
3. Use kmeans++ to solve the clustering problem on each sample.
4. Measure the elapsed time and cost of clustering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

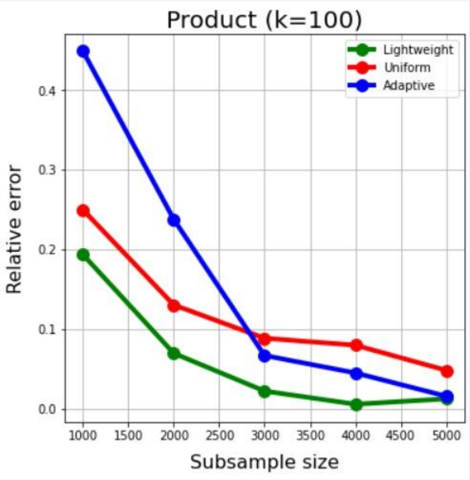
06
Results & Discussions

Results

In this project we compare the three coreset construction methods using relative error as the measurement for the correctness and the run-time comparison.

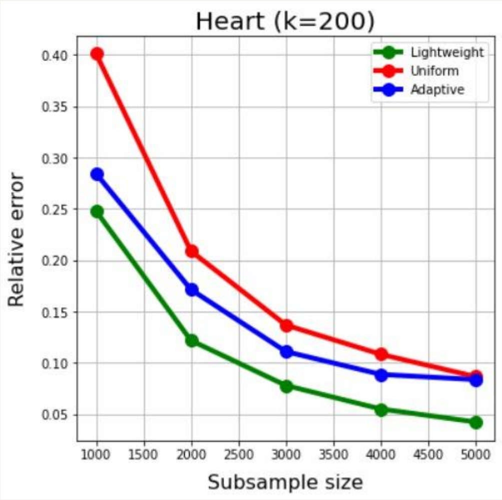


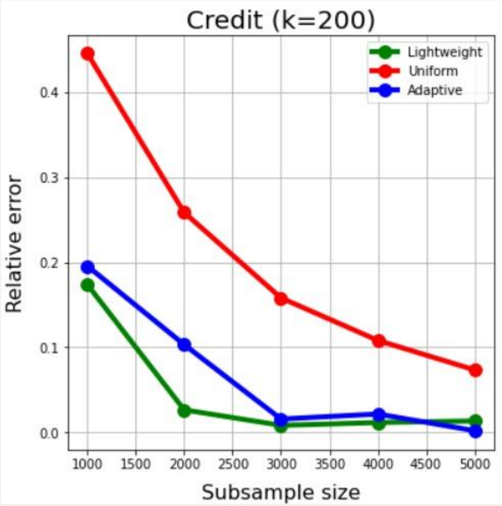


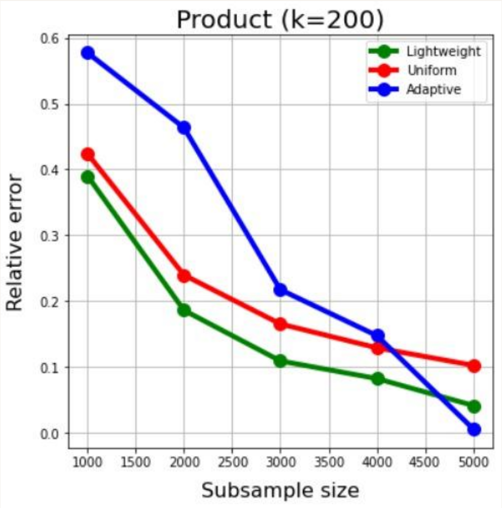


Relative error in relation to subsample size for Uniform, Lightweight and Adaptive Sampling for kmeans++ with k=100

Results







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

Results

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

Results

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

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 Fig. 1 and 2. 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. Lightweight coreset, on the other hand, produces a sample with low error.

Discussions

- 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.

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.

Thank You