Exploring the Power of Coresets for k-Means Clustering

Johns Hopkins University
EN.625.740 Data Mining

Rebecca Kotula

20 November 2023

Introduction

Despite the term “coreset” being coined in 2005, the study and application of coresets seems to be discussed surprisingly little in data science and machine learning circles (Dan Feldman (2020)). This is perhaps because there is not simply one way to define a coreset, as there are different kinds, and thus there is not simply one algorithm to generate a coreset. In general, a coreset of a dataset is some smaller, weighted dataset which yields comparable accuracy when applying the same methods to the coreset as to the original dataset. A coreset is selected and built in such a way that it maintains the necessary characteristics of the original dataset that are relevant to the task at hand. Thus, with negligible change in accuracy, the target task can be performed by training or modeling solely on the coreset instead of the entire large datset.

Coresets were originally studied in the context of the field of computational geometry, but the approaches available at that time were based on computationally expensive methods such as exponential grids (Olivier Bachem (2017a)). Today, many more methods have been presented for selecting a coreset given an input dataset and a cost function we seek to minimize. Finding a coreset has many obvious benefits- most significantly that it can greatly speed up computational cost and time, which is valuable in a variety of scenarios. As “big data” gets larger and larger, methods to improve computational and memory efficiency are not only helpful, they are necessary. For this reason, coresets are also particularly useful in streaming applications, since they involve large amounts of data continuously incoming. Once an accurate and strong coreset has been obtained, streaming data can be easily filtered to determine whether a coreset is updated or not.

Intuitively, coresets make sense. It seems probable that a very large dataset would in fact have some smaller subset that captures the important qualities of the full dataset. However, what seems perhaps less likely is that this same effect could be achieved when starting with a much smaller dataset. Larger datasets can often contain more noise, repetition, and variance in general, just due to their large nature, than small datasets. This begs the question of whether training on coresets of small datasets incurs a higher cost in accuracy than when training on coresets of large datasets. We will explore exactly this question in the following pages.

Project Description

The goal of this project is to explore the effectiveness of coresets on both large and small scale datasets. To do this, we select two datasets, compute coresets (optimized for k-means clustering), and apply k-means clustering to both the original datasets as well as their coresets. We will compare the results both between original datasets and their corresponding coreset, as well as between these two datasets of different sizes. We also explore the relationship between the size of a coreset and the maintained accuracy on the chosen task.

In this paper, we will first provide background information by introducing the basic definition and methodology for constructing a coreset, describing the k-means clustering algorithm, explaining how coresets are constructed for k-means specifically, and finally presenting the datasets used in this project. We will then describe our methods, including steps taken to prepare the data, constructing the actual coresets, and applying the k-means clustering algorithm. Finally, we will present our results, followed by some discussion.

Background

K-Means Clustering

Before we discuss coresets, we will give a simple introduction of the intended function that we'd like to optimize using coresets. In a k-means clustering problem, given a dataset $\mathcal{X}$, we seek to compute a set of k centers. Let us call this set of centers Q, with $|Q| = k$. The selected centers are chosen to optimize

$$cost(\mathcal{X}, Q) = \sum_{x \in \mathcal{X}} \min_{q \in Q} \|x - q\|_2^2,$$

the sum of squared distances. k-means is an iterative problem. First, cluster centers are randomly initialized, given a set value of k. Points are then assigned to clusters according to which cluster center they are closest to. Then, new cluster centers are computed by taking the mean of all the datapoints in each cluster. Points are then reassigned using the new cluster centers, and these steps are repeated until convergence.

Coresets

Coresets are small, potentially weighted summaries of an dataset that provably compete with the accuracy achieved on a specific problem using the larger dataset. By nature, coresets problem-specific because they require a cost function associated with the problem we are trying to solve. If we think about the problem as an optimization problem, as so many problems in machine learning are, we need to construct a coreset that optimizes a specific objective. Thus, a given dataset does not have a singular coreset- the coreset must be computed with a specific problem in mind.

In our explanation, we will assume that the target problem is k-means because that is what is used in this paper, however, keep in mind that coresets can be constructed for many types of problems. Now, given a dataset $\mathcal{X}$ and a solution space of all possible solutions, $\mathcal{Q}$, we seek to optimize a cost function, $cost(\mathcal{X}, Q)$ by finding the optimal solution, Q. For k-means clustering, we want to find a number of clusters k, which minimizes:

$$cost(\mathcal{X}, Q) = \sum_{x \in \mathcal{X}} \mu_{\mathcal{X}}(x) f_Q(x) = \sum_{x \in \mathcal{X}} \mu_{\mathcal{X}}(x) \min_{q \in Q} \|x - q\|_2^2$$

where $f_Q(x) = \min_{q \in Q} \|x - q\|_2^2$ is the squared distance from each point $x \in \mathcal{X}$ to the closest cluster center in Q.

Now, with $\epsilon > 0$, we can formally define a coreset as an $\epsilon - coreset$ of $\mathcal{X}$ if for all $Q \in \mathcal{Q}$:

$$|cost(\mathcal{X}, Q) - cost(\mathcal{C}, Q)| \leq \epsilon * cost(\mathcal{X}, Q)$$

If this holds for all possible solutions $Q \subset \mathcal{Q}$, then $\mathcal{C}$ is a *strong coreset.* If it only holds for the optimal solution, then it is a *weak coreset.*

There are many approaches to constructing a coreset. There are naive approaches, such as uniform sampling, but methods such using importance sampling achieve greater results with smaller coresets. In importance sampling, points are sampled proportionate to their impact on the cost of any given candidate solution. Points are weighted accordingly, and points with low optimum cost are sampled uniformly. Related to importance sampling is sensivity-based sampling. The sensitivity for a point is defined as

$$sens(x) = sup_Q \frac{\min_{q \in Q} \|x - q\|_2^2}{cost_{\mathcal{X}}(Q)}$$

Intuitively, the sensitivity of a point is the maximum share of a point in the cost function. Ideally, a distribution is sampled proportionate to the sensitivity for a point, weighted by their inverse sampling probability.

These are just a few of the many approaches to coreset construction. For example, a very different approach is bounded point movement. Points are moved within specific bounds, and close-together points will be merged and weighted accordingly. However, this project uses an implementation based on the sensitivity framework with importance sampling described by Olivier Bachem (2017a).

Datasets

We explore two datasets in this project. The first is a small dataset, used to demonstrate the coreset selection process, and to prove the results on a simple case. By using this dataset we also seek to determine how effective coresets are on small datasets. The second dataset is much larger, however, due to computational size constraints, it does not approach the size of "big data" for which coresets really have serious cost-saving implications. Nonetheless, it serves to compare the results between two datasets of varying sizes.

Our small dataset is the commonly-used Fisher (1936) Iris dataset. This dataset has 150 observations of 4 features: `Sepal.Length`, `Sepal.Width`, `Petal.Length`, and `Petal.Width`. There are three class labels: `Setosa`, `Versicolor`, and `Virginica`. The anticipated task for this dataset is to classify the species of Iris flower based on its measurements. More specifics, including plots, will presented in the *Methods: Data Preparation* section.

The class distribution in the Iris dataset is:

Var1	Freq
setosa	50
versicolor	50
virginica	50

Based on this, we can anticipate three balanced clusters that we wish to predict in our dataset. We have selected a dataset with class labels in order to simplify the problem space to a scope that can be covered in this paper. In other situations it would be interesting to perform further experiments regarding selecting optimal numbers of clusters; in this paper, we will assume the correct number of (known) clusters has already been selected so that we can address other aspects of the problem.

The larger dataset is a simulated five-dimensional multivariate Gaussian with four distinct classes. Other non-simulated datasets were explored, including an Online Shoppers Purchasing Intention Dataset and a Bank Marketing Dataset from the UCI Machine Learning Repository, but these datasets did not have the attributes necesssary to showcase the features of a K-means coreset that we wish to explore in this paper. We chose to create a simulated data set in order to have a known number of classes, to have a problem that k-means clustering can achieve reasonable success on, and to be able to better visualize the dataset and results. This dataset was generated using the following parameters, with additional Gaussian noise added. Full reproducible dataset construction code can be found at the reference located in the *Appendix*.

```r
class1.mu <- c(10, 5, 7, 9, 20)
class2.mu <- c(9, 3, 4, 5, 15)
class3.mu <- c(3, 2, 1, 4, 9)
class4.mu <- c(5, 5, 5, 5, 5)

n <- 5
A1 <- matrix(runif(n^2)*2-1, ncol=n)
sigma1 <- t(A1) %*% A1
A2 <- matrix(runif(n^2)*4-2, ncol=n)
sigma2 <- t(A2) %*% A2
A3 <- matrix(runif(n^2)*3-2, ncol=n)
sigma3 <- t(A3) %*% A3
A4 <- matrix(runif(n^2)*2-2, ncol=n)
sigma4 <- t(A4) %*% A4
```

There are 16000 observations in this dataset and the class distribution is:

Var1	Freq
a	4000
b	4000
c	4000
d	4000

Both of the chosen datasets have the benefit of even class distribution. This also helps to narrow the problem space, allowing us to focus more closely on our desired aspects. Once again, more specifics and visualizations for this simulated datset will be presented in the *Methods: Data Preparation* section.

t-SNE

We will use t-SNE, or t-distributed Stochastic Neighbor Embedding, to visualize our five-dimensional simulated dataset. We present a quick overview of the algorithm here and note some limitations that will affect what we are and are not able to visualize later in this paper.

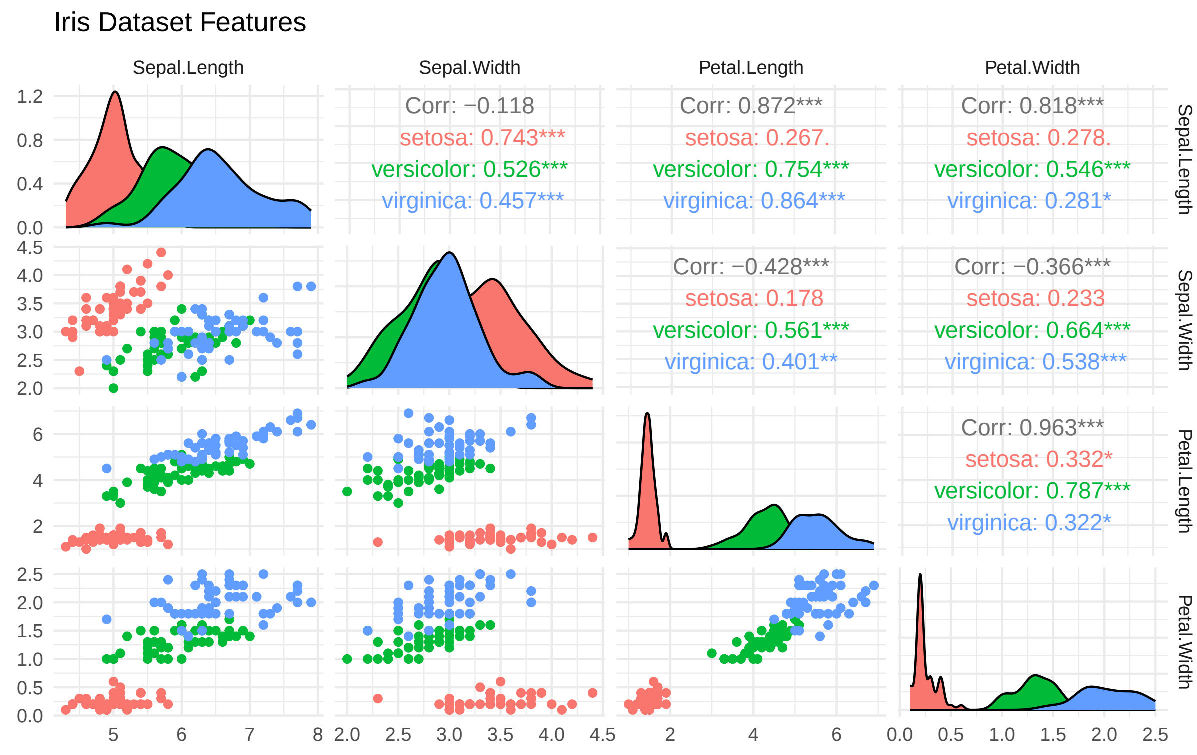
This method was presented by Laurens van der Maaten (2008). It is a technique to visualize high-dimensional data by assigning each datapoint a location on a two or three-dimensional map. It does well both with capturing local relationships of the data and with showing the global structure, such as presence of clusters, which is relevant for our use case. Although the two dimensional maps are created with kowledge of the relationships between the data, it is important to remember that higher-dimensional data cannot be perfectly replicated in a lower-dimensional space. It helps us to visualize our problem, but it is not a perfect representation, and we may not be able to distinguish clusters and patterns that exist in the higher dimensional space.

Another limitation is that once an embedding is computed on a dataset, new points cannot be added to it. It requires either re-computation of the whole dataset, or an additionally trained regression model that is trained on approximating the embedding of the data. In this paper, we will use a few workarounds to visualize our coreset, but this limitation will still come into play.

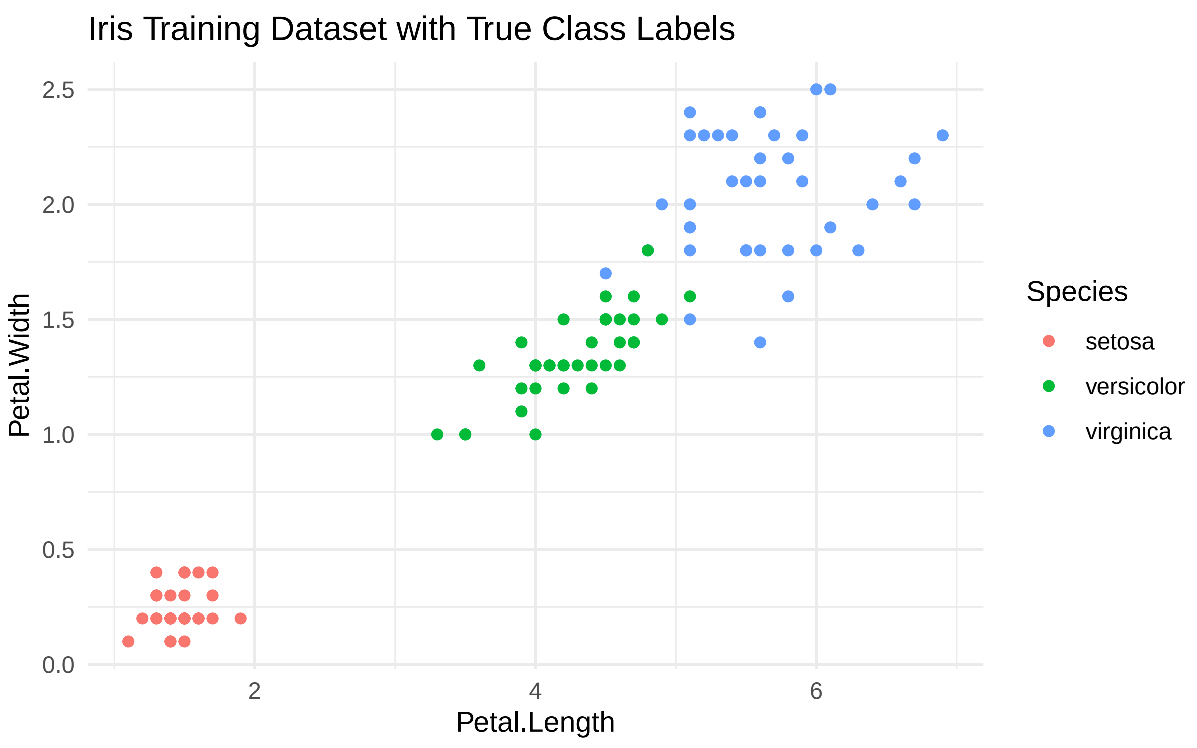
Methods

Data Preparation

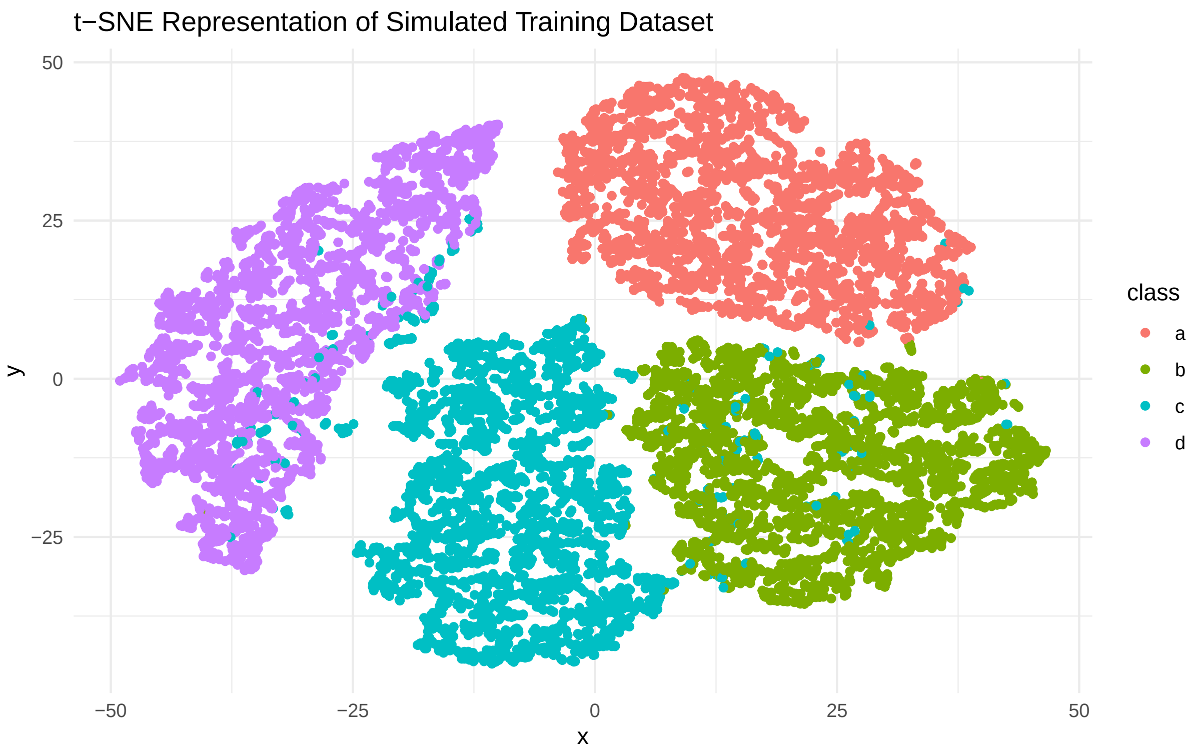
To prepare the Iris dataset, we selected two of the four features to use for clustering in order to have an ideal 2-dimensional visualization for this more basic test case. By visual inspection, the features `Petal.Length` and `Petal.Width` seemed to provide the best class separation into clusters, so we chose these two to perform our analysis on. Additionally, these clusters seemed to be well-formed, so no further pre-processing (such as dealing with outliers) was performed.



The data was split into training and testing sets with a 75/25 split. The training dataset has 112 observations and the testing dataset has 38 observations. Thus, the final selection of the training set can be seen below.

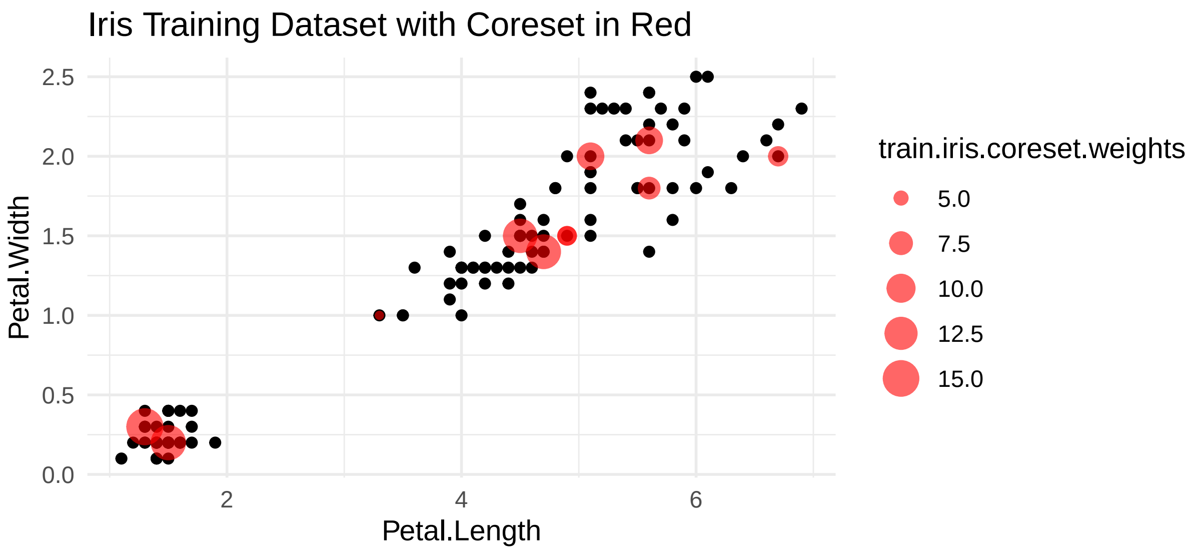


Now we can take a look at our simulated dataset. This data was also split into training and testing sets with a 70/30 split. The training dataset has 12000 observations and the testing dataset has 4000 observations.



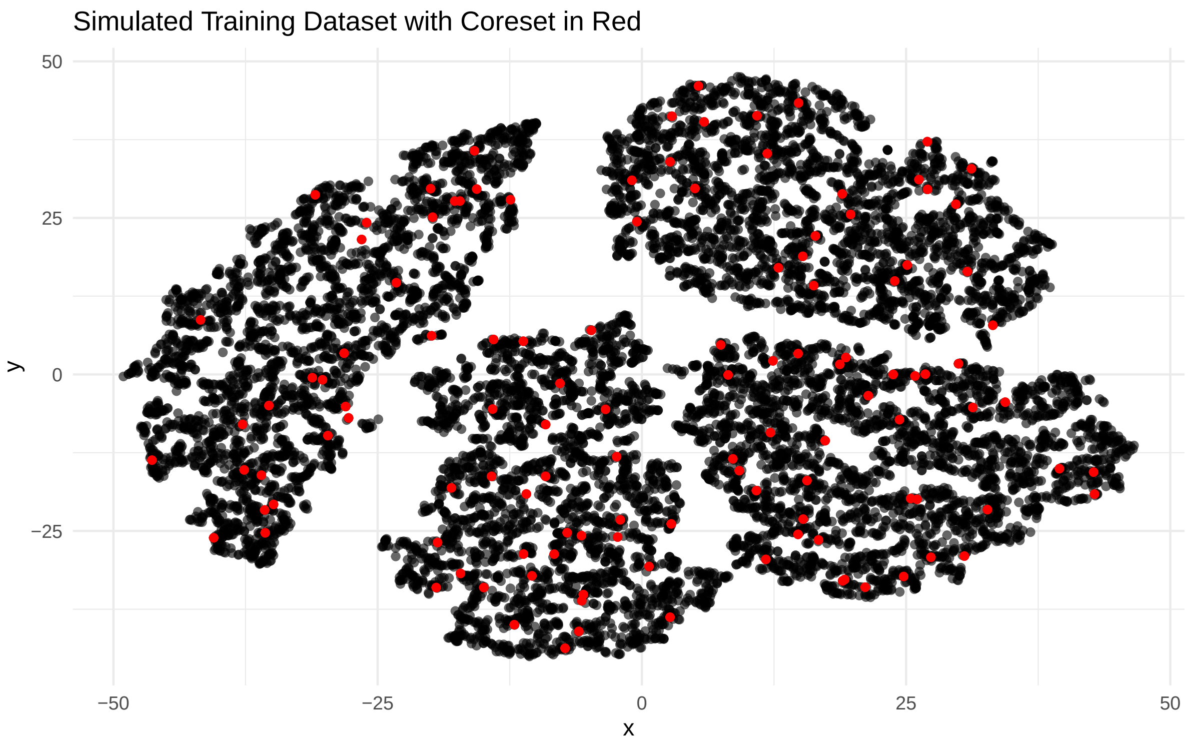
Coreset Construction

Coresets were constructed using the `coresets` package available for Python on PyPI. This implementation is based on the papers by Olivier Bachem (2017a), Olivier Bachem (2017b), Mario Lucic (2018), and Zalán Borsos (2017). As mentioned in the background information, this implementation uses point sensitivity and importance sampling. For the Iris dataset, we report our main results on a coreset that is 10 percent of the size of the original dataset. Across coreset literature, it is common to use coreset of just one percent for large datasets. However, because the Iris dataset is so small, 10 percent is only 11 datapoints. We chose 10 percent to demonstrate the power of the coreset selection without making the coreset absurdly small. Below we can see a depiction of the coreset selected for the Iris training dataset.



The weights here correspond to the importance of each point in the coreset. This generally can be interpreted as the amount of other points that each coreset point is "incorporating".

For our larger simulated dataset, we will report results using a coreset of one percent. This coreset has 120 observations.



Here is where some of the difficulties of t-SNE come in; the coreset was mapped back to the original dataset's projections into the embedding space as well as possible, but it is not a complete representation. However, this visual still helps to give an idea of what the coreset looks like and the amount of data that it covers.

Coreset Size Sweep

In addition to reporting the detailed results of typical-sized coresets for each of our datasets, we perform a sweep over multiple coreset sizes in order to compare the resulting metrics. This table shows the selected coreset sizes (as proportions of the full-sized datasets) used and the resulting number of observations in the coreset for each of our datasets.

size	train.iris	train.sim
0.01	1	120
0.04	4	480
0.05	5	600
0.1	11	1200
0.25	28	3000
original	112	12000

These sizes are computed from the 70% training splits of each of the datasets. We will skip the implementation of 1% on the Iris dataset, because you cannot perform clustering with three clusters on one datapoint.

Clustering

Clustering was performed using the `kmeans` implementation from the `stats` package in R. All clusterings use 25 random initial configurations and report the best one. A custom function was written to assign new data to clusters based on the determined centroids. Code can be found at the indicated location in the *Appendix*.

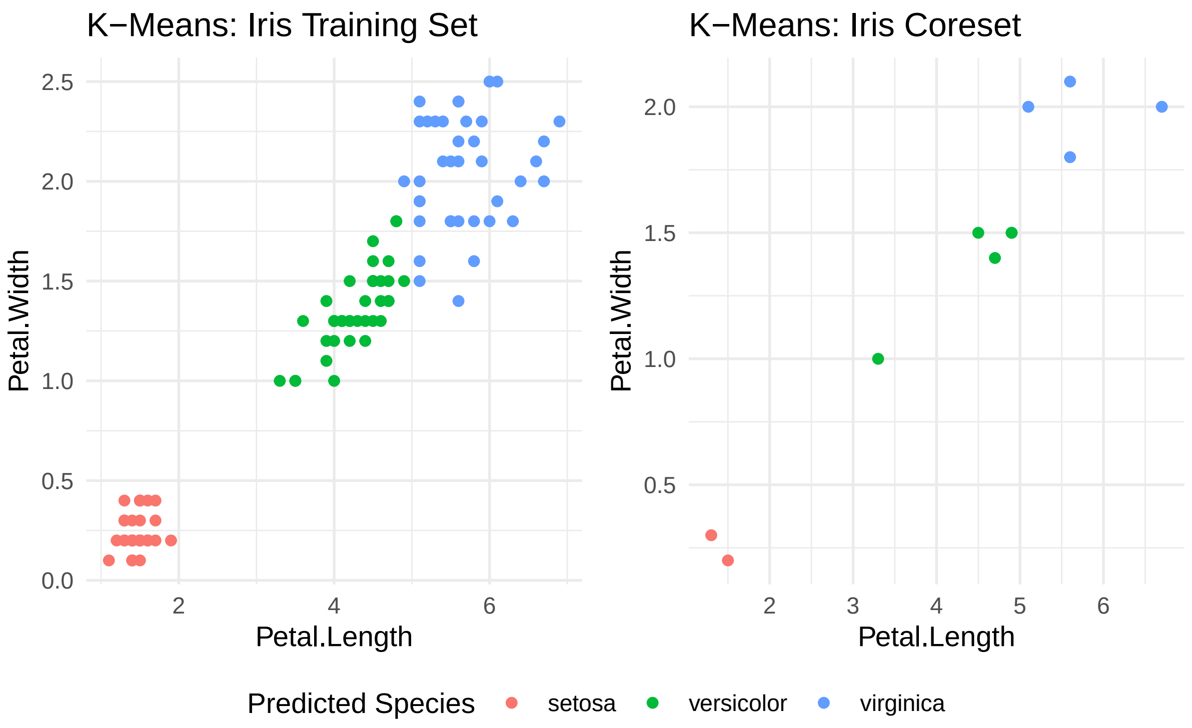
Evaluation Metrics

We evaluate based on two main metrics. Because we have selected datasets in which the class labels are known, we can compute accuracy (as well as confusion matrices) based on the predicted cluster assignments. The other metric we report is the k-means total within-cluster sum of squares. We use the predicted centroids from each model (whether trained on a coreset or full training set) to compute the within-cluster sum of squares on the entire dataset. This gives more insight into how well the k-means centroids fit the dataset as a whole.

Results

Clustering on Original Dataset vs. Coreset

First we will look at the results on the Iris dataset. The k-means clustering assignments on the full training dataset are very accurate.



train.set	test.accuracy	tot.within.ss
full Iris	0.8684211	31.56547
10% Iris coreset	0.8684211	33.36812

We've achieved identical accuracies on the test set, with only a slightly higher total within-cluster sum of squares on the entire dataset using the coreset as the training set. Now we will look more closely at the confusion matrices to see which points are misclassified in the test set.

Table 1: Full training set

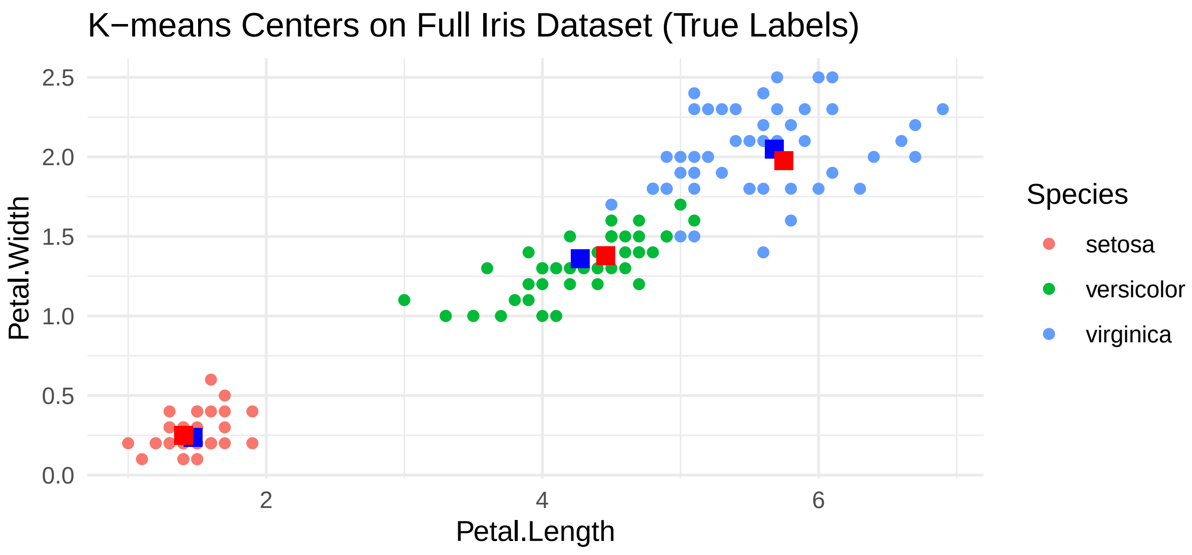
	setosa	versicolor	virginica
setosa	15	0	0
versicolor	0	11	4
virginica	0	1	7

Table 2: Coreset

	setosa	versicolor	virginica
setosa	15	0	0
versicolor	0	12	5
virginica	0	0	6

Our results are extremely similar, and we see there are a few misclassifications between `versicolor` and `virginica`, which we'd expect, since those clusters overlap slightly.

In the plot below, we can see the actual location of the centers predicted by k-means on the full Iris datset with the true labels. The blue squares are the centers predicted by training on the full dataset, and the red are predicted by training on the coreset. Note that they are quite close, indicating once again that we've achieved a similar fit using just the coreset.



Next, let us compare the results on the simulated dataset. We are not able to map these results accurately to the t-SNE embedding, so we will compare metrics without the visualization.

train.set	test.accuracy	tot.within.ss
full sim	0.81775	360816.0
1% sim coreset	0.81875	372697.8

The overall clustering accuracy slightly worse on this dataset as what we achieved on the Iris dataset. But a 1% of the original testing set actually *improves* the accuracy from that of the original dataset. It is quite minimal, but it is still an improvelent. The total within-cluster sum of squares is still quite close as well, but this is actually slightly higher. Now let us look more closely at the confusion matrices.

Table 3: Full training set

	a	b	c	d
a	998	3	3	12
b	4	1011	183	0
c	0	0	647	367
d	0	7	150	615

Table 4: Coreset

	a	b	c	d
a	993	0	2	6
b	9	1012	176	0
c	0	0	685	403
d	0	9	120	585

Again, the results from the coreset model are quite impressive and closely the results from the original model. There is some misclassification mostly involving class `c`, which we would expect based on the t-SNE visualization of the training dataset. Class `c` appears to include more noisy datapoints that cross over into other clusters.

Effect of Coreset Sizes on Clustering

We now analyze the results of different sized coresets of both of our datasets, as described in the *Coreset Size Sweep* section.

Table 5: Iris Dataset Metrics

coreset.size	test.accuracy	tot.w.ss
0.01	–	–
0.04	0.68421052631579	62.26
0.05	0.68421052631579	76.2033333333333
0.1	0.789473684210526	83.9040277777778
0.25	0.921052631578947	32.0692885487528
full	0.868421052631579	31.5654686848366

There are some interesting results here. The Iris dataset achieves the best performance using a 25% coreset, even better than with the full dataset. The Iris results indicate that the pseudo-random seeds, which come into play both in the coreset generation and the k-means fit, have an impact on the results. This is clear because the results from the 10% coreset are slightly lower than those reported earlier in this paper. However, one trend that seems to exist is that there is a significant drop in performance in the coresets smaller than 10%. Still, when we consider that the 4% and 5% coresets are made up of four and five data observations, respectively, the results really are not as atrocious as we might expect.

Table 6: Simulated Dataset Metrics

coreset.size	test.accuracy	tot.w.ss
0.01	0.81875	372697.766744904
0.04	0.8065	368120.681244199
0.05	0.82	363955.333071268
0.1	0.814	362691.341394897
0.25	0.81325	361926.159965202
full	0.81775	360815.991260403

For this larger simulated dataset, we see that the accuracy is extremely close for all coreset sizes. The 5% coreset achieves the highest accuracy, but it is not by very much and, again, this could be attributed to the pseudo-random seed. Additionally, the 1% coreset achieves slightly higher accuracy than the full dataset. The total within-cluster sum of squares is strictly decreasing as we increase the dataset size, however. Overall, it is pretty interesting to note how consistent the accuracy remains, even with coresets as small as 1% of the original dataset.

Conclusion

We have demonstrated how powerful coresets can be to solve k-means problems. And still, this only begins to scratch the surface of the applications of coresets. Our results showed that constructing coresets of large datasets can provide extremely reliable results. The results on the smaller dataset showed a greater drop in accuracy with small coreset sizes. However, our results indicate that coresets still have use for small datasets-one may simply choose to construct a coreset that is a little bit larger in proportion to the full dataset.

Appendix

Code

The code used for this project can be found at https://github.com/beccakotula/exploring_kmeans_coresets.

Bibliography

Dan Feldman. 2020. *Introduction to Core-Sets: An Updated Survey.* Haifa, Israel: Computer Science Department, University of Haifa. https://arxiv.org/abs/2011.09384.

Fisher, R. A. 1936. "The use of multiple measurements in taxonomic problems." Annals of Eugenics. https://doi.org/10.24432/C56C76.

Laurens van der Maaten, Geoffrey Hinton. 2008. *Visualizing Data Using t-SNE.* Tilburg, The Netherlands: Tilburg University. https://www.jmlr.org/papers/volume9/vandermaaten08a/vandermaaten08a.pdf.

Mario Lucic, Andreas Krause, Matthew Faulkner. 2018. *Training Gaussian Mixture Models at Scale via Coresets.* Zurich, Switzerland: Department of Computer Science, ETH Zurich. https://jmlr.org/papers/v18/15-506.html.

Olivier Bachem, Andreas Krause, Mario Lucic. 2017a. *Practical Coreset Constructions for Machine Learning.* Zurich, Switzerland: Department of Computer Science, ETH Zurich. https://arxiv.org/abs/1703.06476.

———. 2017b. *Scalable k-Means Clustering via Lightweight Coresets.* Zurich, Switzerland: Department of Computer Science, ETH Zurich. https://arxiv.org/abs/1702.08248.

Zalán Borsos, Andreas Krause, Olivier Bachem. 2017. *Variational Inference for DPGMM with Coresets.* Zurich, Switzerland: Department of Computer Science, ETH Zurich. http://approximateinference.org/2017/accepted/BorsosEtAl2017.pdf.