Comparison of Clustering Algorithm Performance

Matt Bailey, Neeraj Rajesh, John Roush, Matthew Sinda

Department of Computer Science
Central Michigan University
Mount Pleasant, Michigan 48859
{baile5mj, rajes1n, roush2j, sinda1m}@cmich.edu

Abstract

This paper examines the need for understanding the performance of clustering algorithms, details some different categories of algorithms, and then explores the current state of clustering performance analysis. The paper then discusses an implementation for analyzing clustering performance and then details the results of running such an implementation. Finally, the paper gives an overview of the results of running the implementation and concludes that clustering algorithm performance is highly dependent on the data set which the algorithm is clustering.

1 Introduction

The need to efficiently analysis and process big data is becoming more important. 2007 saw users storing 295 exabytes of digital data [4], and by 2013 the number had grown to 1,200 exabytes [8]. In 2013 alone, computer users stored over 13 exabytes of *new* data [5], which averages over 35 petabytes per day. When data is generated at those rates, it presents challenges, not only in the logistics of storing the data, but also in terms of how to process the data.

Data mining, the process of extracting new knowledge from data, helps solve the problem of how to analyze this wealth of data that is growing on a minute-by-minute basis. Data mining results from a natural evolution of data storage and analysis, and it comprises a combination of statistical and technological methods for learning about data and then making predictions based on what was learned [3]. However, simply choosing a data mining algorithm and applying it to some data set is insufficient; the sheer size of the data being analyzed means we must also consider the performance of the algorithm itself. Consider, for example, an algorithm that runs in $O(c^n)$ time and that is used to process one million records. This algorithm would be useless because it could never process all the records within a reasonable amount of time. Therefore, it would be helpful to know how well some of the common data mining algorithms perform, particularly compared to one another. To that end, we explore some common clustering algorithms, analyze their performance, both in terms of CPU and memory usage, and then look at how accurate their results are.

2 Clustering Algorithms

Clustering involves grouping data objects into groups, or "clusters," where each data object is similar to all other objects in the cluster based on some similarity metric (usually a distance measure) [3]. Clustering algorithms are widely used in data mining because they automatically partition data sets based on a similarity metric and can pretty easily determine which data objects

are related [3]. We chose three types of clustering algorithms to implement and analyze: distance-based, density-based, and probabilistic.

2.1 Distance-Based Algorithms

Distance-based algorithms cluster data based on similarity measured by some distance metric. We looked at two types of distance-based algorithms, k-means and k-medoids. k-Means is a centroid-based partitioning algorithm that clusters data objects based on their distances to the nearest cluster center [3]. It is an iterative process whereby data objects are assigned to clusters, the cluster centers are recomputed based on the mean of all objects in the cluster, and then the process is repeated until the cluster centers no longer change. k-Means, however, is sensitive to outlier objects since outliers distort the mean values of the clusters [3]. k-Medoids attempts to circumvent this problem by choosing a representative data object for each cluster. Instead of calculating the mean of all the data object values, actual data objects are chosen such that they most accurately represent all of the data objects in their respective clusters. The run time for the k-Means algorithm is O(n)[9] while the run time for k-medoids is O(n^2)[7].

2.2 Density-Based Algorithms

Distance-based algorithms build clusters based on some distance from a cluster center, which logically produces clusters that are circular or spherical. However, data objects do not always tend to congregate around centroids. Density-based algorithms, on the other hand, can work around this limitation by producing clusters of arbitrary (i.e. non-spherical) shape [3]. Instead of assigning a data object to the nearest cluster based on its distance from the cluster center, density-based algorithms consider a data object to be either part of a neighborhood (cluster) or noise, depending on surrounding objects. To determine whether a data object resides in a cluster or not, the algorithm looks at either nearness of surrounding objects or uses a density function and noise threshold [3]. This algorithm has a run time of O($nlogn$) if an efficient lookup table is implemented[2].

2.3 Probabilistic Model-Based Algorithms

Probabilistic model-based clustering algorithms assign data points to cluster based on a degree of membership. That is, each data point belongs to its own cluster to some degree, but also belongs to the other clusters to lesser degrees [3]. Probabilistic model-based algorithms determine the probability with which each data object belongs to each cluster (the degree of membership), and then assigns the point to the cluster to which it most likely belongs [3]. This algorithm has a run time of O(n) but is heavily influenced by the number of tuples, attributes and the number of clusters the algorithm needs to find [1].

3 Related Works

Comparing clustering algorithm performance in and of itself does not appear to be a widely studied area, so prior works are relatively sparse. Instead, literature tends to focus on one of two areas: how a newly-developed algorithm compares to an existing algorithm whose problems it was intended to solve or a survey of clustering algorithms which give a general overview of the different types.

[2] discusses both k-means and k-medoids, but the authors only compare their then-new DBSCAN algorithm with CLARANS, a specific k-medoids implementation. For the most part, they

compare DBSCAN to other clustering techniques in terms of how DBSCAN overcomes the weaknesses of other types of algorithms. For example, the authors state that partitioning algorithms such as k-means and k-medoids can only produce convex clusters, but then show that DBSCAN is capable of producing clusters of any shape [2]. Finally, [2] evaluates the performance of DBSCAN, but only as compared to CLARANS, not as part of a larger evaluation of clustering algorithms in general.

k-means was first proposed by in [9]; k-medoids was first proposed by [7]; and c-means, or the EM algorithm was first proposed by [1]. [10] evaluates and compares weighted versions four clustering algorithms: k-means, fuzzy c-means, Gaussian EM, and harmonic means. They first developed highly-mathematical evaluations, then devised a set of experiments that would test their versions of the algorithms in comparison with one another [10]. Similar to [2], the purpose of [10]'s experiments was not to provide a general comparison of clustering algorithms, but to test the weighted versions they developed.

[6] provides a general—but broad and comprehensive—overview of clustering techniques. The authors describe the different classes of clustering algorithms, such as hierarchical, partitional, and fuzzy, and then give high-level overviews of many of the different algorithms in each class. However, [6] limits its discussion of performance to generalities only, observing in one instance, for example, that artificial neural networks perform better than k-means and fuzzy c-means [6]. However, they do not give specific examples of performance comparisons, nor do they devise an experiment to test the differences.

4 Methodology

We were interested in determining how each of our clustering algorithms performed in relation to one another. To do so, we opted to build a Java application which would generate increasingly larger data sets and then run each algorithm on the respective data sets until a predetermined maximum threshold was exceeded. Additionally, we looked at three different types of increases: increase in tuple count, increase in attribute count, and increase in cluster count. We were able to track the memory usage during each run as well as the run time based on start and end times. In addition, we were able to gauge the accuracy of the algorithms since the data was generated with known cluster centers.

4.1 Components

The application comprises the separate clustering algorithms implemented as classes to create pluggable components that can be passed into a testing framework. In addition, there are two utility classes that generate the clustered data sets consisting of both spherical and non-spherical clusters.

4.1.1 Base Class

The base class, `Clustering`, is an abstract class that provides a well-defined interface for plugging into the framework. Each algorithm is then implemented as an extension of `Clustering` so that it can be monitored and analyzed by the testing framework.

4.1.2 Data Set Class

As its name may imply, the data set class, `DataSet`, stores the cluster data for the clustering algorithms to work on. `DataSet` instances are passed into the algorithm implementation classes, which then use the actual data within the instance to produce results. Additionally, `DataSet` provides utility methods for calculating the distance between pairs of tuples. For consistency, we chose to use Euclidean distance for all distance measures.

4.1.3 Data Generator Classes

The data itself presents a challenge when analyzing algorithms that analyze data. There are myriad sources for data sets and the data set itself makes no difference when analyzing performance; as long as all of the algorithms analyze the same data set, we can easily compare how much CPU time and memory each algorithm uses when it runs. However, accuracy poses an entirely different problem: we must determine whether our implementations produce correct results, but we cannot know whether the results are correct without clustering the data beforehand and then comparing the results produced by each of the different implementations.

Ultimately, we decided to reverse-engineer the problem. We created two utility classes that build n-dimensional clusters based on predefined starting data. The first class, `DataGenerator`, creates spherical clusters based upon either specified or randomly-generated cluster centers. The cluster distances and radii are given, and points are generated both within the cluster and as noise.

The second generator class, `generateData`, creates data in non-spherical clusters. Since density-based clustering algorithms are able to cluster non-spherical data, we also needed a way to test the accuracy of our density-based implementation. The `generateData` class creates cluster points along a sine wave, within a specified distance, and then adds noise points to make the data more realistic.

`DataGenerator` and `generateData` allow us to start with known cluster "centers" and generate points around those centers, adding noise to mimic real-world data. Knowing the cluster centers beforehand provides us with a "ground truth" against which we can gauge the accuracy of our algorithm implementations. We can run each algorithm on the data sets and then analyze the results to see how close they are to the known centers.

4.1.4 Algorithm Implementations

There are many different algorithms the perform distance-based, density-based, and probabilistic model-based clustering. We chose to implement four clustering algorithms: k-means and PAM (distance-based), DBSCAN (density-based), and EM (probabilistic-based).

***k*-Means** The k-means implementation is fairly straightforward. It takes initial cluster centers, iterates through each tuple in the data set and assigns it to a cluster based on the closest center, recalculates the centers according to the mean values for each attribute in the tuples, and then starts the process over. This process continues until there are no more changes in the cluster assignments.

The k-means algorithm could be considered one of the simplest algorithms in terms of understanding and implementation. There are no nested `for` loops, so the algorithms runs in $O(n)$ time. The only potential problem is that k-means assigns every tuple to a cluster, so it is sensitive to noise. In addition, extreme outliers have great potential to skew the clusters.

PAM Partitioning Around Medoids (PAM) is an algorithm that performs k-medoids clustering. PAM uses representative objects from the data set. That is, the algorithm tries to find one object per cluster that most accurately represents all of the object in the cluster. PAM requires "seed" objects—initial starting tuples from the data set—which are arbitrarily chosen to represent initial clusters. It then begins an iterative process whereby it replaces representative objects with other objects until the best possible clusters are produced, which is tested by calculating the absolute error.

PAM uses a greedy method by choosing the object with the lowest cost as determined by the average dissimilarity between the object and its cluster representative. Despite using a greedy method, PAM is an inherently inefficient algorithm as it ultimately checks each object against all other objects in an iterative manner; our implementation uses nested `for` loops to mimic this behavior, which causes the algorithm to run in $O(n^2)$ time. However, as an attempt to mitigate this inherent inefficiency, our implementation used a threshold value that limited the number of object to be chosen as potential representative objects.

DBSCAN Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a density-based clustering algorithm. DBSCAN uses two user-specified parameters, epsilon (ϵ) and minimum points ($MinPts$), to find regions of dense "neighborhoods" of data objects, which are then considered clusters. Density is a function of the radius of the neighborhood, specified by ϵ, and a neighborhood is defined by $MinPts$. That is, for each data object in the data set, DBSCAN looks at how many other objects are within a radius of ϵ, and if that value is greater than or equal to $MinPts$, then the object is considered a core object. From there, DBSCAN finds small dense regions by connecting core objects with respect to ϵ and $MinPts$, and finally it builds large dense regions by connecting objects within a radius of ϵ of already-connected objects.

The biggest limiting factor with our DBSCAN implementation proved to be with regard to the the data set implementation. We chose to store the tuples in a single-dimensional array since we needed a common data structure for each implementation to use, and a single-dimensional array seemed the simplest. However, since the DBSCAN algorithm requires first iterating through each tuple, then through each core tuple, then through each connected core tuple, our implementation ended up using nested `for` loops, which resulted in a loop structure that would run in $O(n^2)$ time.

EM Expectation-maximization (EM) is a technique for producing fuzzy clusters and probabilistic model-based clusters. EM starts with randomly-selected objects as potential centers and then iteratively works in two steps: the expectation step and the maximization step. The expectation step iterates through each data object, calculates the distance to each of the potential centers, and then calculates a partition matrix containing values that indicate to what degree each object belongs to a cluster centered at each of the potential centers. The partition matrix, containing membership weights, is then normalized so the sum of degrees for each object is 1. Next, the maximization step uses the partition matrix to recalculate the centroids, by calculating an optimal center and then finding the data object closest to that optimal center. The process repeats until the clusters stop changing or until the change is below some predefined threshold.

EM runs until there are no more changes. Despite running in $O(n)$ time, there is potential for the algorithm not to converge, but to end up with increasingly smaller changes, which requires a threshold to determine when to stop running the algorithm. One challenge we also faced was that the floating point error inherent in all CPUs introduces inaccuracies, so "no more changes" automatically becomes a matter of "sufficiently small changes." Either way, setting a threshold values was sufficient to overcome both of these problems.

4.1.5 Testing Framework

The testing framework, `AlgorithmTests`, accepts data generator class instances and uses those to generate larger and larger sets of n-dimensional tuples until either a specified maximum number of runs has occurred or until a specified maximum time threshold has passed. The generated data are passed into instances of the algorithm implementation classes, which cluster the data. The framework then tracks the performance of each algorithm in terms of total processing time and memory usage and outputs the results to a text file. Additionally, `AlgorithmTests` contains methods for testing performance based on increasing numbers of tuples, increasing numbers of dimensions, and increasing numbers of clusters.

5 Results

We looked at the results of the clustering algorithms both in terms of how accurately each algorithm calculated clusters compared to what we were expecting (i.e. the "ground truth") and in terms of performance (both run time and memory usage). Section 5.2 gives the results of the generated clusters on varying types of data sets, Section 5.3 discusses the clustering results of the data set used for testing, and Section 5.4 gives the absolute performance results.

5.1 Algorithm Implementation

In table 1, a comparison can be made between the run time of our algorithm implementation and the original proposed algorithm. It can be seen that we matched the run times for all algorithms except for DBSCAN. This mismatch was due to not implementing an efficient lookup method for this algorithm. This would have required significant work on the common data structure used throughout the program and significantly over complicated the program. If this was for a commercial program, all attempts would be made to improve the efficiency of the DBSCAN algorithm. However, as we are only comparing the algorithm against other clustering algorithms, the difference in run times for our purpose is acceptable.

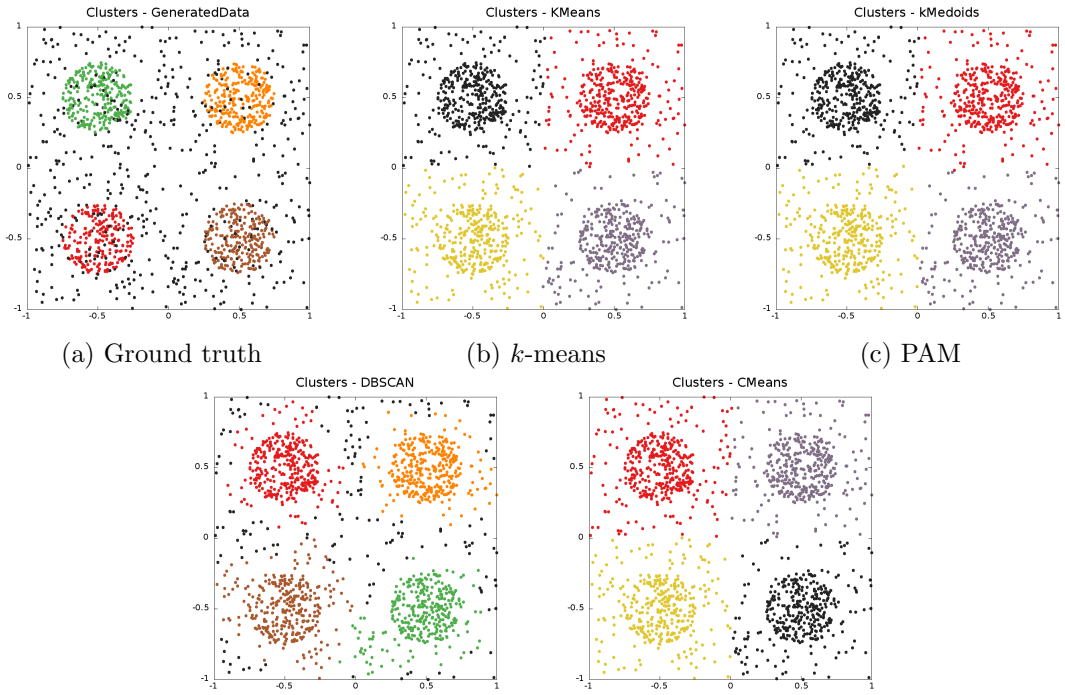
	Implementation	Original
PAM	$O(n^2)$	$O(n^2)$
k-means	$O(n)$	$O(n)$
EM	$O(n)$	$O(n)$
DBSCAN	$O(n)$	$O(nlogn)$

Table 1: Comparison of Run Times

5.2 Clustering Results

Figure 1 illustrates the results of clustering a basic data set with random noise points. As we can see in Figure 1d, DBSCAN produced the most accurate results, successfully distinguishing noise from cluster points, and coming closest to matching the ground truth (Figure 1a). On the other hand, k-means, PAM, and EM all produced relatively similar results, clustering all points without regard to noise.

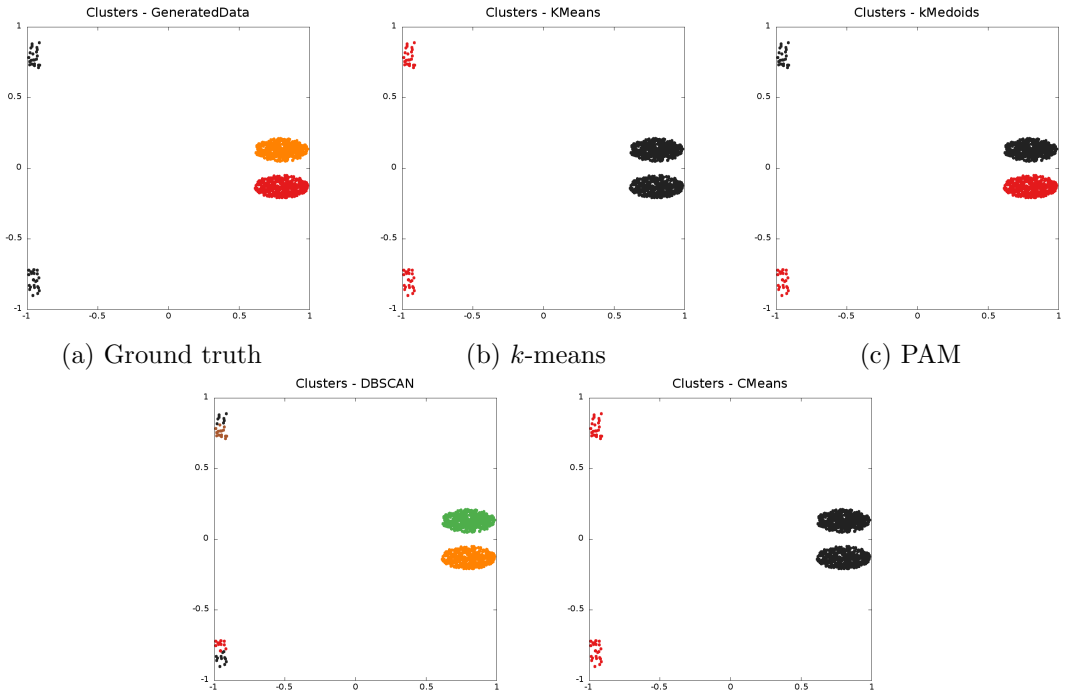
Figure 2 shows a data set still containing spherical clusters, but with extreme outlying points rather than random noise. In this instance, the accuracy of the algorithms is not quite as clear. As expected, k-means, PAM, and EM clustered all points, with k-means and EM producing the



(a) Ground truth (b) k-means (c) PAM

(d) DBSCAN (e) EM

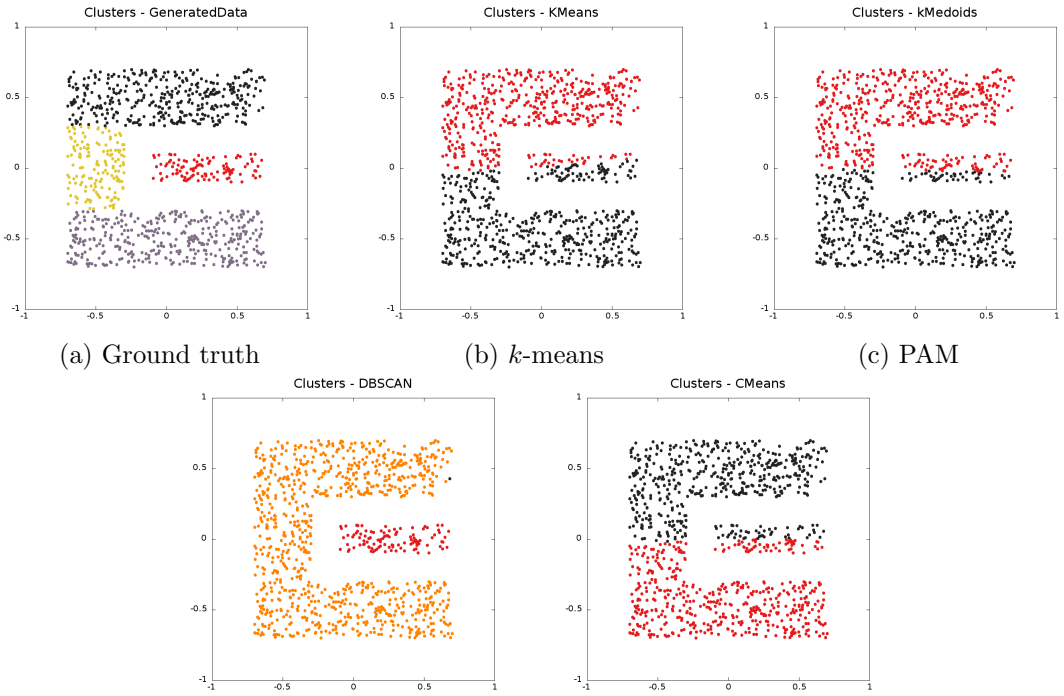
Figure 1: Comparison of clustering results on a data set containing spherical clusters with random noise.



(a) Ground truth (b) k-means (c) PAM

(d) DBSCAN (e) EM

Figure 2: Comparison of clustering results on a data set containing spherical clusters and extreme noise points.



(a) Ground truth (b) k-means (c) PAM

(d) DBSCAN (e) EM

Figure 3: Comparison of clustering results on a data set containing a concave cluster.

same results. PAM and DBSCAN both successfully distinguished the clusters, however PAM also included the outlier points in the clusters and DBSCAN did not correctly categorize all outliers, but instead clustered some of the outliers.

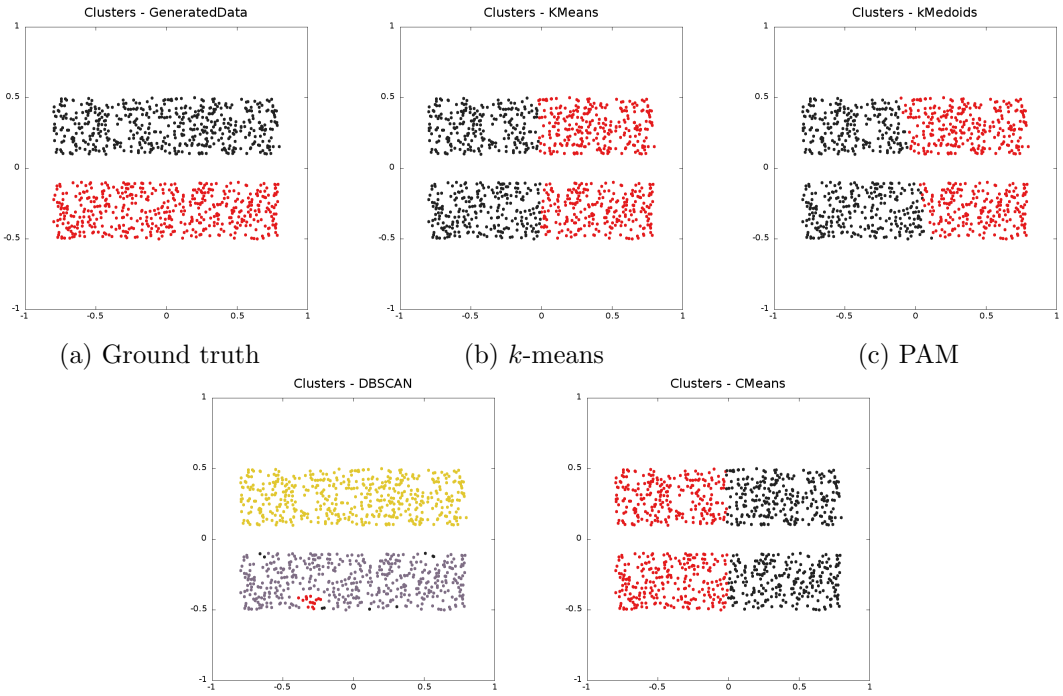
We can also see in Figure 3 that DBSCAN performs the best when picking out a concave cluster. Although the ground truth (Figure 3a) shows four clusters, the results of DBSCAN (Figure 3d) perhaps make more sense, visually. In looking at the data points without consideration of actual cluster assignments, most people would likely see the points as being in two clusters as illustrated in Figure 3d.

In a data set with oblong clusters as shown in Figure 4, DBSCAN once again performs the best. Although Figure 4d shows that it did categorize some points as outliers, DBSCAN did, for the most part, produce the expected clusters illustrated in 4a.

DBSCAN ended up being the best performer overall when clustering different types of data sets. Although it did not produce 100% accurate results, it was relatively accurate, particularly with its ability to distinguish noise points from clusters points. On the other hand, k-means, PAM, and EM all produced essentially the same results regardless of the data set, so we must look at their memory and CPU performance to see if there are any distinguishing characteristics.

5.3 Actual Data Set

Figure 5 shows the resulting cluster assignments from the data set that was used for performance testing. The data was generated to be clustered around nine centers as shown in Figure 5a (note that cluster C4 contains the random noise, not clustered points). Figure 5d shows that DBSCAN produced clusters that most closely resemble the expected results, although it failed to find all nine clusters.



(a) Ground truth (b) k-means (c) PAM

(d) DBSCAN (e) EM

Figure 4: Comparison of clustering results on a data set containing two oblong clusters.

Figures 5b and 5e show that neither k-means nor EM identified noise points, which is normal since those algorithms will cluster all points with no consideration to noise. Interestingly, both k-means and EM discovered clusters in areas where the original data had no cluster points, as shown by the lower-right corners of Figures 5b and 5e versus Figure 5a.

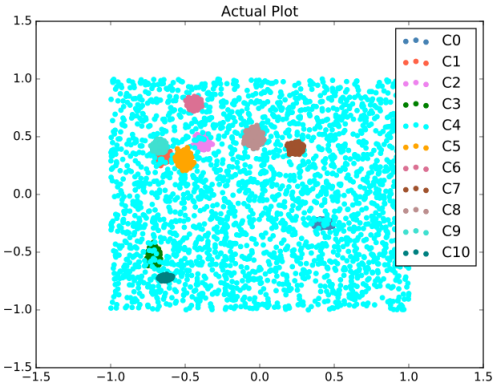
PAM was the one outlier in terms of accurately finding clusters. It found only a single cluster, which contained all tuples in the data set. As a result, we will need to further investigate our PAM implementation to determine if it is a fault of the algorithm itself or with our implementation.

5.4 Performance

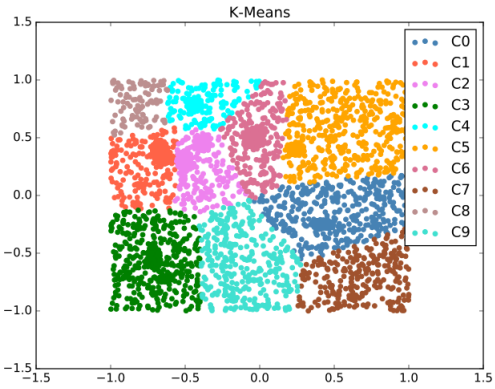
Figure 6 shows how our implementations performed as the number of tuples in the data set increased. All four algorithms performed about the same in terms of memory usage: memory consumption increased linearly as the number of tuples increased. In terms of maximum memory usage, though, k-means outperformed the others by consuming no more than 40 KB of memory. By contrast, EM was the clear loser at a maximum of 11 MB.

We can also see some difference in the algorithms in terms of time performance. Figures 6b and 6c show that PAM and DBSCAN run in $O(n^c)$ time for some power c, which is expected since both run in $O(n^2)$ time. On the other hand, the functions describing the running times for k-means and EM are not immediately clear, although further testing may reveal their performance trends. Figures 6a and 6d show that time to cluster did not steadily increase, but instead tended to have relative ups and downs with an overall upward trend. Additionally, Figure 6d shows a dramatic increase in run time as the data set increased from about 6,000 to 7,000 tuples, followed by a dramatic relative decrease in run time as the data set increased from about 7,000 to 8,000 tuples.

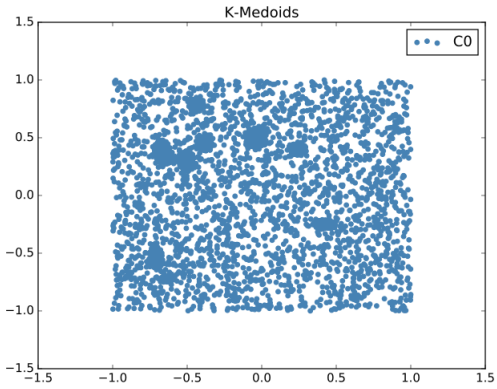
Figure 7 illustrates that the number of attributes in each tuple had a dramatic effect on the



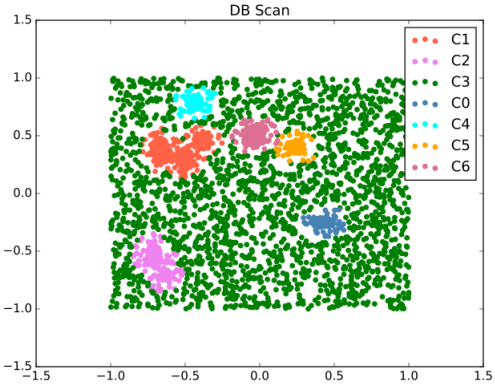
(a) Original data



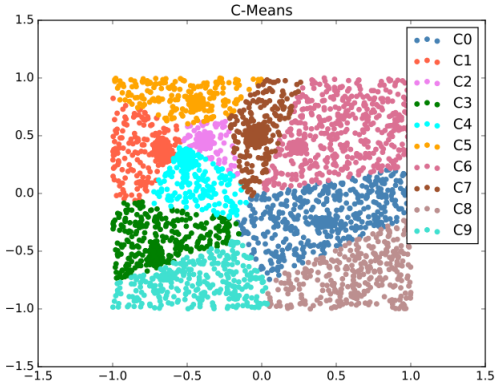
(b) k-means.



(c) PAM

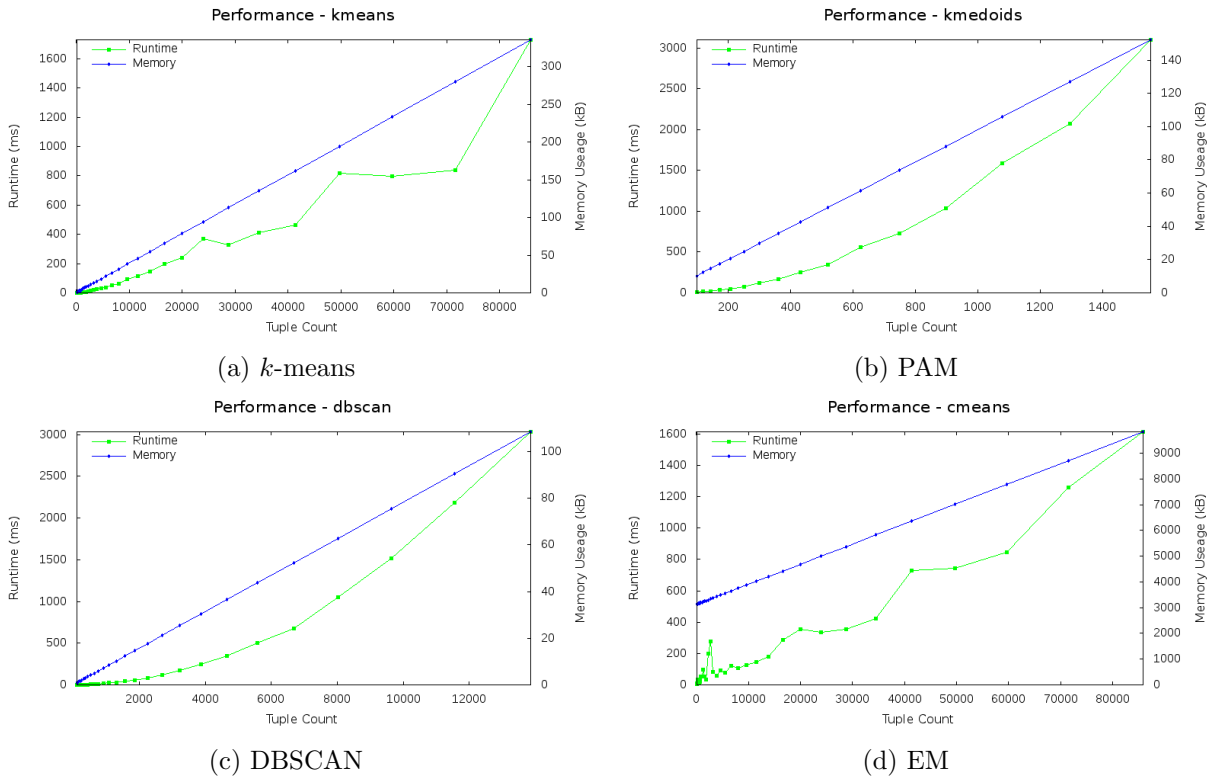


(d) DBSCAN



(e) EM

Figure 5: Expected clusters (as produced by the data generator) and clusters produced by the algorithm implementations.



(a) k-means

(b) PAM

(c) DBSCAN

(d) EM

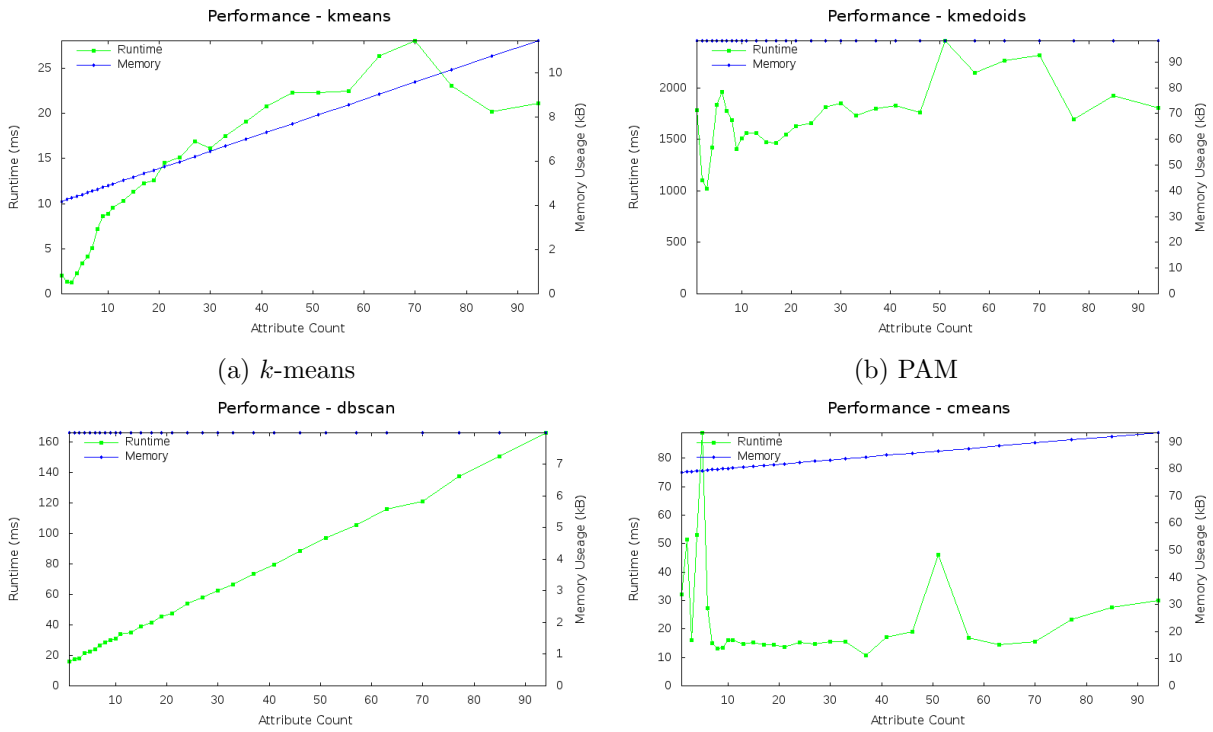
Figure 6: Performance of algorithm implementations based on increasing number of tuples.

performance of some of the algorithms. In terms of memory usage, Figures 7a, 7b, and 7d show that k-means, PAM, and EM were all relatively stable and that the usage for all three increased linearly as the number of attributes increased. DBSCAN, however, exhibited ups and downs in its memory usage as shown in Figure 7c, which could potentially be a result of Java's garbage collector dynamically allocating and deallocating memory.

Figures 7a, 7b, and 7c show that k-means, PAM, and DBSCAN saw a relatively linear increase in running time as the number of attributes increased. EM's running time, as seen in Figure 7d, spiked near the beginning of the test, and then remained relatively constant once the number of attributes increased from about seven onward. Overall, though, PAM appeared to be the most performant in terms of both time and memory.

Figure 8 shows performance as the cluster count increases. Figures 8a and 8b show that both k-means and PAM saw a relatively linear increase in both processing time and memory usage. EM also saw a linear increase in memory usage as seen if Figure 8d. On the other hand, DBSCAN's memory performance was much the same as when the number of attributes were increasing. Again, this could be because of dynamic memory (de)allocation during the program's run.

In terms of time performance, both DBSCAN and EM were relatively unstable as Figures 8c and 8d show. However, DBSCAN's running time did not exhibit quite as much relative variation as EM's did. Overall, k-means and PAM appear to be tied for best performance, with a slight edge going to k-means.



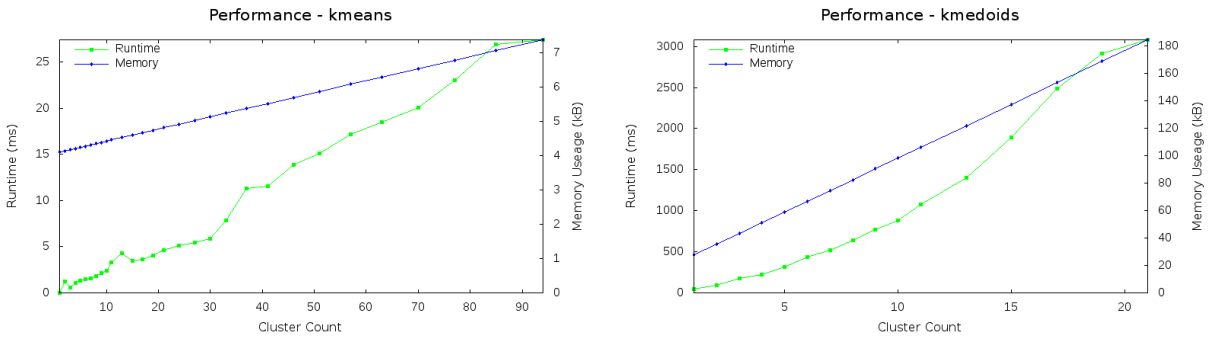
(a) k-means

(b) PAM

(c) DBSCAN

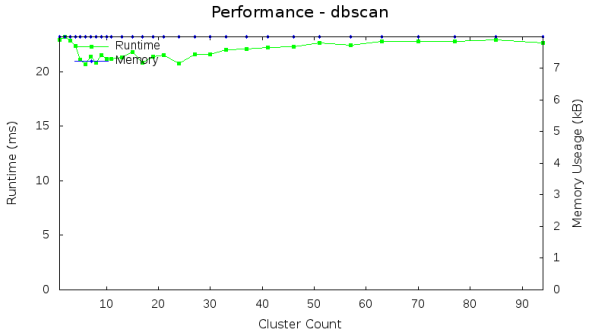
(d) EM

Figure 7: Performance of algorithm implementations based on increasing number of attributes per tuple.

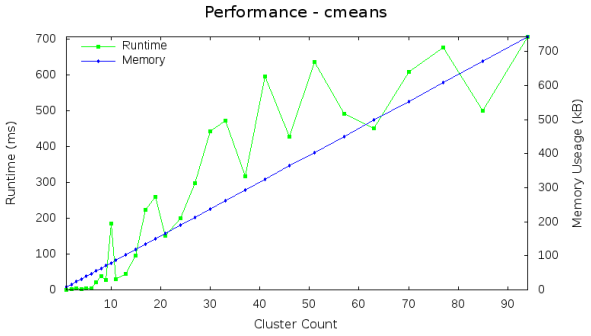


(a) k-means

(b) PAM



(c) DBSCAN



(d) EM

Figure 8: Performance of algorithm implementations based on increasing number clusters to find.

6 Conclusion

Ultimately, no one algorithm stands out as being the best. Instead, each algorithm seems best-suited for different applications. Going in to the project, we were aware that k-means, k-medoids, and fuzzy c-means were all sensitive to outliers, which is apparent in the cluster plots. However, we did not know that each algorithm (including DBSCAN) would be better suited to different types of data sets than the others. For example, DBSCAN tended to perform worse than the others, but was most accurate at creating clusters that match those of the original data set. In the end, we can conclude that clustering analysis should involve clustering the data using more than one algorithm and then choosing the one that produces the most accurate results.

References

[1] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 38(1):1–38, December 1977.

[2] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[3] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques*. Elsevier, Amsterdam, third edition, 2012.

[4] Martin Hilbert and Priscila López. The World's Technological Capacity to Store, Communicate, and Compute Information. *Science*, 332(6025):60–65, April 2011.

[5] H. V. Jagadish, Johannes Gehrke, Alexandros Labrinidis, Yannis Papakonstantinou, Jignesh M. Patel, Raghu Ramakrishnan, and Cyrus Shahabi. Big data and its technical challenges. *Commun. ACM*, 57(7):86–94, July 2014.

[6] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, September 1999.

[7] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups of Data: An Introduction to Cluster Analysis*. John Wiley and Sons, Hoboken, 1990.

[8] Juha Lehikoinen and Ville Koistinen. In big data we trust? *Interactions*, 21(5):38–41, September 2014.

[9] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129–137, March 1982.

[10] R. Nock and F. Nielsen. On weighting clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1223–1235, Aug 2006.