Sampling Within k-Means Algorithm to Cluster Large Datasets

Team Members: Jeremy Bejarano,[1] Koushiki Bose,[2] Tyler Brannan,[3] Anita Thomas[4]

Faculty Mentors: Kofi Adragni[5] and Nagaraj K. Neerchal[5]

Client: George Ostrouchov[6]

Technical Report HPCF–2011–12, `www.umbc.edu/hpcf` > Publications

Abstract

Due to current data collection technology, our ability to gather data has surpassed our ability to analyze it. In particular, k-means, one of the simplest and fastest clustering algorithms, is ill-equipped to handle extremely large datasets on even the most powerful machines. Our new algorithm uses a sample from a dataset to decrease runtime by reducing the amount of data analyzed. We perform a simulation study to compare our sampling based k-means to the standard k-means algorithm by analyzing both the speed and accuracy of the two methods. Results show that our algorithm is significantly more efficient than the existing algorithm with comparable accuracy.

Key words. k-means, clustering large datasets, sample size, tolerance and confidence intervals.

AMS subject classifications (2010). 65C20, 62H30, 62D05, 62F25.

1 Introduction

Scientific researchers are currently collecting vast amounts of data that need analysis. For example, NASA's Earth Science Data and Information System Project collects data from seven satellites called the Afternoon Constellation. Each satellite simultaneously measures atmospheric, oceanic, and chemical readings from different parts of the world. In total, they transmit three Terabytes (1 Terabyte = 1024 Gigabytes) of data per day to Earth. Datasets such as these are difficult to move or analyze.

Analysts often use clustering algorithms to group data points with similar attributes. Though various clustering algorithms exist, our research focuses on Lloyd's k-means algorithm which iterates until the algorithm converges. It must calculate $N \cdot k$ distances per iteration, where N is the total number of data points which are to be assigned to k clusters. These distance calculations are time-intensive especially with multi-dimensional data. Dr. George Ostrouchov suggested that sampling could ease this burden. The key challenge

[1]Department of Mathematics, Brigham Young University

[2]Department of Applied Mathematics, Brown University

[3]Department of Mathematics, North Carolina State University

[4]Department of Applied Mathematics, Illinois Institute of Technology

[5]Department of Mathematics and Statistics, University of Maryland, Baltimore County

[6]Oak Ridge National Laboratory

d	Best	Average
1	0.0000	0.0004
2	0.0000	0.0001
3	0.0001	1.2570
4	0.0000	0.0062

Table 1.1: Absolute difference between the *standard* and *sampler* accuracy values (in percentages).

was to find a sample size that was large enough to yield accurate results but small enough to outperform the standard k-means' runtime. To counter this problem, we introduced a sampling function within the k-means algorithm. The sample size n is calculated after every iteration. A random sample of size n is then used in the next iteration of the algorithm. Once the algorithm converges on our sample, the entire dataset is classified into clusters, and the k centers of the clusters are calculated.

We perform a simulation study to compare the efficiency and accuracy of our algorithm, henceforth referred to as the *Sampler*, to the results obtained by simply running the standard k-means, henceforth called the *Standard* on the entire dataset. Both best and average accuracy on a scale from 0 to 100 were examined for each algorithm after each simulation's twenty trials. Table 1.1 lists the absolute difference between the two algorithms' accuracy values for the number of dimensions d. The algorithms have almost identical accuracies. Let t_1 be the *Standard*'s total runtime and t_2 be the *Sampler*'s total runtime. Figure 1.1 plots the ratio t_1/t_2 versus number of dimensions. Thus, the sampler was t_1/t_2 times faster than the standard as indicated by the vertical coordinate. Therefore, we conclude that the sampler algorithm matches the accuracy of k-means while significantly decreasing runtime.

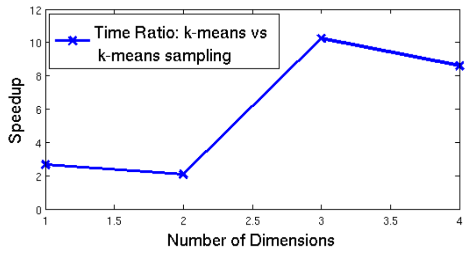

Figure 1.1: Ratio of total times of twenty trials ($t1/t2$).

The report is organized as follows. Section 2 introduces this problem in the context of both k-means algorithms and hypothesizes sampling as a solution. Section 3 discusses the implementation of sampling within the k-means algorithm and how sample size is determined. Section 4 presents the results of the simulation study for datasets having different cluster patterns and dimensions, examining the accuracy and runtimes of our methods in comparison to the standard k-means algorithm found in `R` [4].

2 The Algorithm and Incorporation of Sampling

2.1 The k-Means Algorithm

The k-means algorithm [2] is a heuristic method of cluster analysis: the grouping of data points to illustrate underlying similarities. Given a set of observations $X = (x_1, x_2, \dots, x_N)$ and a set of k means, $\bar{X} = (\bar{x}_1, \bar{x}_2, \dots, \bar{x}_k)$, this algorithm partitions N observations into k clusters, $S_1, S_2, \dots, S_k$, with $k \leq N$. Clusters are determined based on each obervation's Euclidean distance from each mean. The distance of every observation in X from each mean in $\bar{X}$ is computed. Observations are subsequently classified to the cluster S_j whose center they are the closest, thereby minimizing the within-cluster sums of squares. Forming clusters in this manner is known as the assignment step. The update step involves computing new centers by finding the means of all the observations in a particular cluster. The assignment and update steps are performed iteratively until convergence occurs. Convergence is said to occur when, in a particular iteration, no data point is reclassified to a new cluster.

Typically, k-means is run multiple times, and the"best classification" is recorded, since a poor choice of initial centers can result in incorrect classification and may even prevent the algorithm from converging. "Best classification" is determined by comparing the within-cluster sum of squares for each run; the run having the minimum within-cluster sum-of-squares is identified as the "best classification".

2.2 Sampling Within k-Means

Our algorithm, the *Sampler*, is a modification of Lloyd's k-means algorithm [1] found in the software `R` [4]. Initially, the assignment step runs on only a minute sample of the dataset, as discussed in Section 3.3. Once these points are classified, the algorithm proceeds to the update step. Next, the sample is updated using a function named `block_sample`. The purpose of this function is to determine the size of the sample needed for the next iteration. Section 3.1 discusses the `block_sample` function in detail. There are two difficulties in randomly picking a sample from the dataset in each iteration: choosing random points from a large dataset is time-consuming and is complicated by the need to sample without replacement. This is especially difficult as the target datasets that we intend to use are extremely large. This enlarges what are referred to as NUMA (Non-Uniform Memory Access) issues. If we are to analyze only a small sample of the whole dataset randomly scattered across all the data, the *Sampler* will lose considerable time efficiency as it jumps around to read these data points. To minimize runtime, we estimate the maximum sample size n^* that the algorithm will need. Additionally, memory is saved since a new matrix containing only the first n^* data points of a random permutation of the original dataset is stored, rather than all N points. We estimate the maximum size

$$n^* = k \left[\frac{1}{N} + \left(\frac{w}{2z^*} \right)^2 \right]^{-1}, \tag{2.1}$$

where the values of w and z^* rely on a desired width of a confidence interval for the cluster means. In constructing this confidence interval, we take the standard deviation to be one, which is the maximum since the data has been centered and sphered. Additionally, to keep our calculations consistent with Section 3.1, we apply a finite population corrector, taking N as our population cluster size, so as to overestimate rather than underestimate the maximum sample size.

Thus, only n^* points are randomly chosen from our data and stored in a new dataset where $n^* \ll N$. When a sample of size n is needed, the first n points of this new dataset are chosen. This prevents repeated NUMA issues by storing the part of the dataset that the algorithm uses contiguously in memory. Hence we only need to access consecutive points in our dataset, reducing our runtime.

When the algorithm converges, i.e., no points change their classification in a particular iteration, we have a subset of the population that has been clustered into k clusters. On the basis of the centers obtained, all N points in the dataset are now classified, and k new centers are computed one last time based on the clusters. These k centers and k clusters are the final output of our sampling algorithm.

3 Calculating Sample Size

3.1 Determination of Sample Size within Each Iteration

To cluster all the observations in our population, we need to find the centers (means) of each cluster using k-means. In order to do this, we construct a confidence interval of a chosen width around the means of each of the clusters in the sample, and estimate the sample size needed for that confidence interval. For each of the k clusters, we use the sample mean of each cluster $\bar{x}_j$, (for $j = 1, 2, \ldots, k$), as an estimate for the means for the population clusters. For large samples, by the central limit theorem we can say that for each of the clusters, the sample mean follows an approximate normal distribution. Therefore, given a confidence level C, the confidence interval is of the form

$$\bar{x}_j \pm z^* \frac{\sigma_j}{\sqrt{n_j}} \qquad j = 1, \ldots, k$$

where σ_j is the population standard deviation for cluster j, n_j is the size of the sample needed from cluster j, and z^* is the upper $(1 - C)/2$ critical value for the standard normal distribution. The value of z^* is determined within the algorithm depending on the confidence interval input by the user. In our problem, we are drawing samples without replacement, i.e., in every iteration, the additional points we choose do not belong to our existing sample set. We also consider each sample point as coming from a cluster in the population. However, we do not know the sizes of the clusters in our population. Because of this fact, our sample size may not remain small in comparison with the population size, and therefore, we need to make use of a *finite population correction* (fpc) [3, pages 27–28]. Since we are trying to apply a confidence interval to the mean of every cluster, we need to know the population size corresponding to every cluster. We call these population cluster sizes N_j. Since we do not know the exact clusters in our population, we can estimate N_j by examining the quantity

$$\frac{nc_j}{n},$$

which is the proportion of sample points that were classified into cluster j in the previous sample. Here nc_j refers to the number of points of the previous sample that were classified into cluster j by the previous iteration, and n is the previous sample size. Then our population cluster sizes become

$$N_j = \frac{nc_j}{n} N. \tag{3.1}$$

Hence, our confidence interval takes the form

$$\bar{x}_j \pm z^* \frac{\sigma_j}{\sqrt{n_j}} \sqrt{\frac{N_j - n_j}{N_j}},$$

which can be simplified to

$$\bar{x}_j \pm z^* \sigma_j \sqrt{\frac{1}{n_j} - \frac{1}{N_j}}.$$

If the chosen width of the above confidence interval is to be w, we get the following equation

$$w = 2\left(z^* \sigma_j \sqrt{\frac{1}{n_j} - \frac{1}{N_j}} \right). \tag{3.2}$$

Solving the above expression for n_j we have

$$n_j = \left[\frac{1}{N_j} + \left(\frac{w}{2z^* \sigma_j} \right)^2 \right]^{-1}. \tag{3.3}$$

Once the sample size for each cluster is determined, the sample size n to be used in the next iteration is:

$$n = \sum_{j=1}^{k} n_j. \tag{3.4}$$

3.2 Estimation of variance

Within each iteration of our *Sampler* algorithm, we recalculate the sample size for the desired confidence interval. As shown in Equation (3.3), this calculation requires an estimate of σ_j for each cluster S_j for $j = 1, 2, \ldots, k$. Appealing to the central limit theorem, we treat the points within each cluster $x \in S_j$ with $x \in \mathbb{R}^d$ as normally distributed along each dimension, centered at the true cluster centers $\mu_j \in \mathbb{R}^d$ with standard deviation $\Sigma_j \in \mathbb{R}^{d \times d}$. Also, we assume that each dimension is independent and thus $\Sigma_j = diag\left((\sigma_{1,j})^2, \ldots, (\sigma_{d,j})^2\right)$. Therefore,

$$S_j^i \sim \mathcal{N}\left(\mu_j^i, (\sigma_{i,j})^2\right). \tag{3.5}$$

Note that throughout this section we use superscripts to denote a projection onto the ith component dimension: thus $x \in \mathbb{R}^d$ but $x^i \in \mathbb{R}$ for $i = 1, \ldots, d$ and S_j^i as the set (here the j-th cluster) whose elements have been projected onto the the i-th dimension. In our algorithm we chose to estimate $\sigma_{i,j}$ using the rough estimator

$$6\hat{\sigma}_{i,j} = \max_{x \in S_j} x^i - \min_{x \in S_j} x^i. \tag{3.6}$$

This then leaves us with $d \cdot k$ estimates of standard deviations. Each $\hat{\sigma}_{i,j}$ for $j = 1, 2, \ldots, k$ and $i = 1, \ldots, d$ could potentially be used to calculate a sample size in Equation (3.3), however, we wish only to produce one estimate $\hat{\sigma}_j$ for each cluster. We thus define the estimate of standard deviation (only in the sense that we use it to determine sample size) for each cluster as the largest estimate among each dimension:

$$\hat{\sigma}_j := \max_i \hat{\sigma}_{i,j}. \tag{3.7}$$

3.3 Initial Sample Size

In the first iteration of our algorithm, the *Sampler* clusters a minute sample from the dataset. Our `block_sample` function has yet to run; thus we need an initial sample size. That size is important since `block_sample` uses information from the previous iteration. Specifically, if $p_1, p_2, \ldots, p_k$ are the true proportions of the population data points for the k clusters, then we want our initial sample to have approximately $p_1, p_2, \ldots, p_k$ proportions of points from the corresponding cluster.

To obtain a representative first sample, we can construct a confidence interval for the proportions [5, pages 411–415]. Choosing a confidence level of C and a width of w, the sample size is determined below. The value z^* in Equations (3.8) and (3.9) represents the critical value for the $(1-C)/2$ upper of the standard normal distribution. We use the worst-case scenario of $p = 0.5$ to determine the variance, so that we get a conservative estimate of the sample size required.

$$w = 2\left(z^* \sqrt{\frac{p(1-p)}{n}}\right) \tag{3.8}$$

$$n = \left(\frac{2z^* \sqrt{p(1-p)}}{w}\right)^2 \tag{3.9}$$

Solving for n with $w = 0.06$ and $C = 95\%$, we get $n \approx 1000$. Our code takes this number as given. Note that if the size of the dataset is less than 1000, the *Sampler* will simply use the entire dataset, no different than the *Standard*.

4 Simulation Study

4.1 Accuracy and Efficiency of k-Means Sampler

In examining the accuracy and efficiency of the two algorithms, we perform studies to compare the classifications, cluster centers, and runtimes.

Accuracy (Acc) is measured by the percentage of points correctly classified. The Euclidean norm between the true centers μ_j of our generated dataset and the centers $\bar{x}_j$ determined by the algorithms is used as a measure of error

$$\sum_{j=1}^{k} \|\bar{x}_j - \mu_j\| = Err. \tag{4.1}$$

The time taken is simply the wall clock time observed for each of the two algorithms. Note that for the *Sampler*, the time taken to randomize the data as discussed in Section 2.2 is included. Randomization is essential for this algorithm in case the data has underlying patterns. Hence, this time cannot be ignored when making comparisons. Additionally, the time taken to center and sphere the data is included for both algorithms.

4.2 Data Generation

For our study we generate test data as follows. We generate the points within each cluster $x \in S_j$ with $x \in \mathbb{R}^d$ from normal distributions.We generate points across every dimension i such that they are normally distributed along each dimension, centered at the true cluster centers $\mu_j \in \mathbb{R}^d$ with standard deviation $\Sigma_j \in \mathbb{R}^{d\times d}$ (we use the same notation as in Section 3.2). Each dimension is generated independently and thus $\Sigma_j = diag\left((\sigma_{1,j})^2, \ldots, (\sigma_{d,j})^2\right)$. Again,

$$S_j^i \sim \mathcal{N}\left(\mu_j^i, (\sigma_{i,j})^2\right). \tag{4.2}$$

For our study, all points generated by the parameters $(\mu_j^i, \sigma_{i,j})$ are said to have a true classification of j.

4.3 Results

Our first study examined four datasets, of one, two, three, and four dimensions. We used a 95% confidence level and a width of 0.01 in all our studies. The cluster sizes were all equal in this study. Results for $N = 119,603,200$, $k = 4$, with each algorithm run 20 times on the same dataset, are given in Tables 4.1 and 4.2. The maximum sample size n^* calculated for all our datasets is $613,845$. Notice that this is only 0.5132% of the entire dataset, and assists in the speedup of our algorithm. For the 20 trials, Table 4.1 gives the best results obtained from each of the two algorithms, for each dataset. Table 4.2 lists the averages and standard errors (s.e.) of the three statistics reported. In each of the 20 trials, both the algorithms were provided with identical centers, with different centers across runs. "Best results" refer to the particular run of the algorithm that gave the minimum within-cluster sum of squares. The

(*a*)One Dimension	Acc	Err	Time
k-Means	99.0679	0.0235	1113
k-Means Sampler	99.0679	0.02277	419
(*b*)Two Dimensions	Acc	Err	Time
k-Means	98.7610	0.0331	1619
k-Means Sampler	98.7610	0.0331	770
(*c*)Three Dimensions	Acc	Err	Time
k-Means	99.8563	0.0044	11787
k-Means Sampler	99.8562	0.0044	1151
(*d*)Four Dimensions	Acc	Err	Time
k-Means	99.9719	0.0039	15729
k-Means Sampler	99.9719	0.0039	1830

Table 4.1: Best results obtained from generated data: comparison with true clusters.

third column in Table 4.1 gives the total time taken for all the 20 runs. This is significant since it allows us to compare the times taken by both the algorithms in achieving similar accuracies and errors. It is seen that they both have nearly identical errors and accuracies, but the *Sampler* significantly and consistently outperforms the *Standard* in terms of time.

We notice a marked difference in results as we proceed to higher dimensions. In the case of three-dimensional data, we notice that both the algorithms do not perform as well on average, seen in Table 4.2 (c). However, if we examine the best results, we achieve 99.9% accuracy and $Err = 0.0044$ for both the algorithms. Thus, eventually, both algorithms give good results but show a large difference in time. The *Sampler* takes only $1,151$ seconds for 20 trials, compared to $11,787$ seconds for the *Standard* as seen in Table 4.1 (c). Therefore, for three dimensions, it is considerably more efficient to use the *Sampler*. Similar conclusions can be drawn from four-dimensional data in Tables 4.1 (d) and 4.2 (d).

Our second study was on datasets with different types of clusters. These datasets also had $119,603,200$ points. Although the data was generated in a similar way compared to the first study, the cluster sizes were not equal, along with other minor changes in parameters. Results for these datasets are very similar to the ones observed in Tables 4.1 and 4.2. Once again, best results, accuracies, errors, means and standard errors are all very similar for both the algorithms. Likewise, we also notice a significant decrease in time when the *Sampler* is used.

Figure 4.1 summarizes the time results for the data used in first study. It plots the times taken for all 20 trials for both the algorithms, against the number of dimensions. Notice that although both the algorithms increase in time with dimensions, this increase is minimal in case of k-means sampler, showing an approximately linear increase.

Results for two of the 20 trials from three-dimensional data are given in Table 4.5. Table 4.5 (a) gives the results for a particular run when both the algorithms performed well. Table 4.5 (b) gives results for a particular run when both the algorithms performed poorly. These results illustrate two vital points.

(a)One Dimension	Acc	s.e.(Acc)	Err	s.e.(Err)	Time	s.e.(Time)
k-Means	99.0679	0.0000	0.0235	0.0000	55.6679	2.2271
k-Means Sampler	99.0675	0.0001	0.0259	0.0008	20.9617	0.1414
(b)Two Dimensions	Acc	s.e.(Acc)	Err	s.e.(Err)	Time	s.e.(Time)
k-Means	98.7610	0.0000	0.0331	0.0000	80.9680	1.4953
k-Means Sampler	98.7611	0.0000	0.0332	0.0000	38.5109	0.1475
(c)Three Dimensions	Acc	s.e.(Acc)	Err	s.e.(Err)	Time	s.e.(Time)
k-Means	76.1653	5.7822	10.0762	2.3218	589.3905	112.7390
k-Means Sampler	77.4223	5.4655	9.9661	2.3000	57.5484	0.2734
(c)Four Dimensions	Acc	s.e.(Acc)	Err	s.e.(Err)	Time	s.e.(Time)
k-Means	69.9848	6.9862	12.9450	2.9797	786.4350	148.6328
k-Means Sampler	69.9910	6.9855	12.9360	2.9776	91.4829	2.5904

Table 4.2: Summary of results obtained from generated data: comparison with true clusters.

(a)Two Dimension	Acc	Err	Time
k-Means	98.7328	0.1371	1873
k-Means Sampler	98.7324	0.1376	778
(b)Three Dimensions	Acc	Err	Time
k-Means	99.9092	0.0275	8455
k-Means Sampler	99.9092	0.0276	1161

Table 4.3: Best results obtained from second version of generated data: comparison with true clusters.

(a)Two Dimension	Acc	s.e.(Acc)	Err	s.e.(Err)	Time	s.e.(Time)
k-Means	98.7328	0.0000	0.1371	0.0000	93.6284	3.0807
k-Means Sampler	98.7329	0.0004	0.1369	0.0004	38.8944	0.0805
(b)Three Dimensions	Acc	s.e.(Acc)	Err	s.e.(Err)	Time	s.e.(Time)
k-Means	88.8786	4.4288	13.9884	4.6640	422.7588	102.2657
k-Means Sampler	88.8703	4.4272	13.9645	4.6713	58.0578	0.3901

Table 4.4: Summary of results obtained from second version of generated data: comparison with true clusters.

(a)Correct Clusters	Accuracy	Error	Time
k-Means	99.8564	0.0044	95.0131
k-Means Sampler	99.8564	0.0044	56.6900
(b)Incorrect Clusters	Accuracy	Error	Time
k-Means	37.5989	21.3402	1090.0576
k-Means Sampler	37.7884	21.4123	57.1936

Table 4.5: Example differences in runtime.

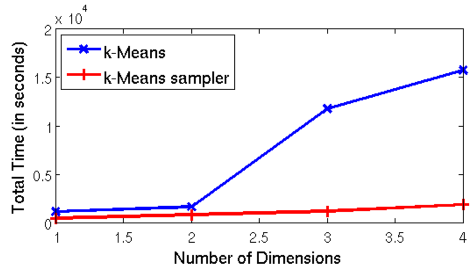

Figure 4.1: Variation in time for best results across dimensions.

First, for all 120 trials with our 6 datasets, both algorithms either work extremely well or quite poorly. In none of our trials did one algorithm do well while the other did poorly. When we ran our studies, we noticed that the poor performance can be attributed to poor random initial centers, e.g, starting with three centers from the same true cluster. Since both the algorithms begin with identical centers in every trial (but different centers between trials), if our algorithm does choose a sample that is representative of the entire dataset, it follows that both algorithms should perform similarly. Thus, the fact that both algorithms either succeed or fail together for each trial gives further credence to the assertion that our algorithm can match k-means in terms of accuracy.

Second, Table 4.5 alerts the reader as to why the *Sampler* has drastically better times in three and four dimensions. If the algorithms fail, they both usually reach the maximum number of iterations, 250 for our studies, or a number close to that. However, the *Sampler* runs approximately 250 times on no more than $n^* = 613,845$ points while the *Standard* uses all $119,603,200$ points. Therefore, the *Standard* takes significantly more time to get a wrong answer. As mentioned in Section 2.1, k-means is usually run mutltiple times due to the possibility of poor random initial centers. Thus, this time issue is quite important as an analyst may run k-means five times to ensure the best results. But, if the *Standard* fails even once, the runtime will be significantly longer than for the *Sampler*.

5 Conclusion

Further work on this project might include a more comprehensive study both on more varied test datasets as well as on real weather datasets. This is especially important considering that this preliminary study was performed on rather tame datasets. Also, these datasets should analyze the performance of the algorithm on varied values of k. Lastly, this paper showed that the algorithm was accurate for relatively low sample sizes. We would like to analyze this further to see how accurate the algorithm is for even lower sample sizes. We could find the lowest sample sizes, by manipulating width and confidence level, for which the algorithm would be acceptably accurate.

In order for our algorithm to be a success, it needs to meet two benchmarks: match the accuracy of the standard k-means algorithm and significantly reduce runtime. Both goals are accomplished for all six datasets analyzed. However, on datasets of three and four dimension, as the data becomes more difficult to cluster, both algorithms fail to obtain the correct classifications on some trials. Nevertheless, our algorithm consistently matches the performance of the standard algorithm while becoming remarkably more efficient with time. Therefore, we conclude that analysts can use our algorithm, expecting accurate results in considerably less time.

Acknowledgments

This research was conducted during Summer 2011 in the REU Site: Interdisciplinary Program in High Performance Computing (`www.umbc.edu/hpcreu`) in the UMBC Department of Mathematics and Statistics, funded by the National Science Foundation (grant no. DMS–0851749). This program is also supported by UMBC, the Department of Mathematics and Statistics, the Center for Interdisciplinary Research and Consulting (CIRC), and the UMBC High Performance Computing Facility (HPCF). The computational hardware in HPCF (`www.umbc.edu/hpcf`) is partially funded by the National Science Foundation through the MRI program (grant no. CNS–0821258) and the SCREMS program (grant no. DMS–0821311), with additional substantial support from UMBC.

References

[1] S. P. Lloyd, *Least squares quantization in PCM*, IEEE Transactions on Information Theory, 28 (1982), pp. 128–137.

[2] J. MacQueen, *Some methods for classification and analysis of multivariate observations*, in Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, L. M. Le Cam and J. Neyman, eds., vol. 1, Berkeley, CA: University of California Press, 1967, pp. 281–297.

[3] N. K. Neerchal and S. P. Millard, *Environmental Statistics*, CRC Press LLC, 2001.

[4] R Development Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2011. ISBN 3-900051-07-0, http://www.R-project.org/.

[5] D. D. Wackerly, W. Mendenhall, and R. L. Scheaffer, *Mathematical Statistics with Applications*, Cengage Learning, seventh ed., 2008.