Acceleration of K-Means and Related Clustering Algorithms

Steven J. Phillips

AT&T Labs-Research
180 Park Avenue, Florham Park, NJ 07932
phillips@research.att.com

Abstract. This paper describes two simple modification of K-means and related algorithms for clustering, that improve the running time without changing the output. The two resulting algorithms are called COMPARE-MEANS and SORT-MEANS. The time for an iteration of K-means is reduced from $O(ndk)$, where n is the number of data points, k the number of clusters and d the dimension, to $O(nd\gamma + k^2 d + k^2 \log k)$ for SORT-MEANS. Here $\gamma \leq k$ is the average over all points p of the number of means that are no more than twice as far as p is from the mean p was assigned to in the previous iteration. COMPARE-MEANS performs a similar number of distance calculations as SORT-MEANS, and is faster when the number of means is very large. Both modifications are extremely simple, and could easily be added to existing clustering implementations.
We investigate the empirical performance of the algorithms on three datasets drawn from practical applications. As a primary test case, we use the Isodata variant of K-means on a sample of 2.3 million 6-dimensional points drawn from a Landsat-7 satellite image. For this dataset, γ quickly drops to less than $\log_2 k$, and the running time decreases accordingly. For example, a run with $k = 100$ drops from an hour and a half to sixteen minutes for COMPARE-MEANS and six and a half minutes for SORT-MEANS. Further experiments show similar improvements on datasets derived from a forestry application and from the analysis of BGP updates in an IP network.

1 The K-Means Algorithm

Let P be a set of n points in R^d, and let k be a non-negative integer. Suppose we wish to cluster the points of P, partitioning them into k classes $C_1, \ldots, C_k$ so that each class consists of points that are close together. K-means [7] is a widely used algorithm for clustering, and sees frequent application in such diverse fields as astronomy, remote sensing, and speech recognition.

The basic operation of K-means is the iteration of the following two steps. Let $\mu_1, \ldots, \mu_k$ be the means of the classes found in the previous iteration.

1. Assign each point $p \in P$ to the class C_j that minimizes $d(p, \mu_j)$.
2. Recalculate the means: for each $j \in \{1 \ldots k\}$, set μ_j to be the mean of the points assigned to C_j in Step 1.

D. Mount and C. Stein (Eds.): ALENEX 2002, LNCS 2409, pp. 166–177, 2002.

K-means can be started with any set of initial means, and ends when no point changes class in Step 1. It is easy to show that the distortion ($\sum_j \sum_{p \in C_j} d(p, \mu_j)^2$) is monotonically decreasing, and that K-means ends with a local minimum of the distortion.

1.1 The Isodata Algorithm

There are a number of heuristics for helping K-means avoid bad local minima, such as when some means get "orphaned", having no points assigned to them. Because much of the data used in this paper is satellite imagery, we use a heuristic that is extensively used in such remote sensing applications, called Isodata (Iterative Self-Organizing Data Analysis Technique) [10,6].

Isodata specifies an initial set of means and a set of rules for discarding, splitting, and merging classes. In each dimension i, let μ_i be the mean and σ_i the standard deviation of the points. The initial means are spaced at even intervals on the line between $(\mu_1 - \sigma_1, \ldots, \mu_k - \sigma_k)$ and $(\mu_1 + \sigma_1, \ldots, \mu_k + \sigma_k)$. When a class size falls below `minimum_class_size` the class is discarded. If the number of classes is less than k, and some class has standard deviation higher than `maximum_standard_deviation` in some dimension i, then the class is split in two, with the new means a standard deviation apart in dimension i. There are rules for tie-breaking (when more than one class has high standard deviation), and the parameter `minimum_distance` determines when nearby classes should be merged. For further details, see Jensen [6].

1.2 Contributions of This Paper

The ideas described here reduce the running time of K-means, without changing the output. The resulting algorithms are COMPARE-MEANS, which takes advantage of the clustering obtained in each iteration to avoid making unnecessary comparisons between data points and cluster means in the next iteration, and SORT-MEANS, which makes a similar number of comparisons, while improving the running time further by reducing overhead.

The degree of acceleration improves with the quality of the clustering; this is quantified in Section 2.3. For our algorithms to be useful, not only must the data set have clusters, but K-means must also be finding a reasonable clustering. We therefore chose to study data sets drawn from applications where K-means (or the Isodata variant) see frequent and successful application. The results of the study are contained in Section 4. We chose not to do experiments on synthetic data, as there is a huge range of data distributions that could be used, and it is not clear which, if any, would accurately model data that would be observed in practice.

2 Accelerating K-Means and Isodata

2.1 Acceleration by Comparing Means

In Step 1 of the K-means algorithm, each point p is compared against each mean μ_j, resulting in $O(nk)$ comparisons and a running time of $O(ndk)$ per iteration. The algorithm COMPARE-MEANS uses a simple approach to avoid many comparisons that are provably unnecessary.

Consider a point $p \in P$ and consider two means μ_i and μ_j. Using the triangle inequality, we have $d(\mu_i, \mu_j) \leq d(p, \mu_i) + d(p, \mu_j)$, so

$$d(p, \mu_j) \geq d(\mu_i, \mu_j) - d(p, \mu_i).$$

Therefore if we know that $d(\mu_i, \mu_j) \geq 2d(p, \mu_i)$ we can conclude that $d(p, \mu_j) \geq d(p, \mu_i)$ without having to calculate $d(p, \mu_j)$.

Algorithm COMPARE-MEANS simply precomputes the distance $d(\mu_i, \mu_j)$ for each pair of means before each iteration. Then before comparing a point p to a mean μ_j, we perform the above test using the closest known mean to p. Each iteration is as follows:

1a. Calculate all inter-mean distances $D[i][j] = d(\mu_i, \mu_j)$.
1b. For each point $p \in P$, let c be the class p was assigned to in the previous iteration (by default $c = 1$ for the first iteration), and do the following steps. At the end, p is assigned to class `minClass`.

```
minDist = d(p, mu[c]);
minClass = c;
for (i=1; i<=k; i++) {
    if (D[i][minClass]) >= 2*minDist || i==c)
       continue;
    dist = d(p, mu[i]);
    if (dist<minDist) {
       minDist = dist;
       minClass = i;
    }
}
```

Note that we initialize `minDist` using the mean p was assigned to in the previous iteration. The is because for many datasets, in all but the first few iterations most points don't change class, so the last assigned mean is frequently the closest mean.

2.2 Acceleration by Sorting Means

A further speedup is obtained by first sorting the means in order of increasing distance from each mean. An iteration of SORT-MEANS is as follows.

1a. Calculate all inter-mean distances $D[i][j] = d(\mu_i, \mu_j)$, then construct the $k \times k$ array M, in which row i is a permutation of $1 \ldots k$, representing the classes in increasing order of distance of their means from μ_i.

1b. For each point $p \in P$, let c be the class p was assigned to in the previous iteration, and do the following steps. At the end, p is assigned to class `minClass`.

```
inClassDist = d(p, mu[c]);
minDist = inClassDist;
minClass = c;
for (i=2; i<=k; i++) {
    theClass = M[c][i];
    if (D[c][theClass]) >= 2*inClassDist
       break;
    dist = d(p, mu[theClass]);
    if (dist<minDist) {
       minDist = dist;
       minClass = theClass;
    }
}
```

In words, we compare point p against the means in increasing order of distance from the mean μ_c that p was assigned to in the previous iteration. If we reach a mean that is far enough from μ_c, we know we can skip all the remaining means and continue on to the next point. In this way, Sort-means avoids the overhead of looping through all the means.

Note that we could instead use the expression `(inClassDist + minDist)` in place of `2*inClassDist`, which would reduce the number of comparisons for some points that change class during the iteration. We chose not to do this in the implementation described in this paper, as the simpler expression allows us to easily avoid taking square roots in the distance calculations (recording squared distances in `D[][]`, `inClassDist` and `minDist`).

For the first iteration of Sort-means we may have an initial assignment of points to means (see for example the discussion of bootstrapping at the end of Section 4.1). If not, we can arbitrarily use $c = 1$ for all points, and unless otherwise noted, this is done for simplicity in the experiments below. A slight reduction in the number of comparisons can be obtained by instead using Compare-means for the first iteration; the reduction is significant when we have a good initial set of means, but we don't have a good initial assignment of points to those means.

2.3 Analysis

In an iteration of K-means, let γ be the average over all points p of the number of means that are no more than twice as far as p is from the mean p was assigned to in the previous iteration. The number of comparisons that Sort-means makes

in the iteration is at most $n\gamma$, rather than nk. Note that $\gamma \leq k$. One can construct datasets for which γ is close to k, however for datasets on which K-means finds a good clustering, with most points being much closer to their assigned mean than to most other means, we can expect γ to be much smaller than k. The running time of an iteration is $O(nd\gamma + k^2 d + k^2 \log k)$, where the second and third terms are the time taken to compare and sort the means.

The number of comparisons made by COMPARE-MEANS is harder to characterize. If a point p does not change class in the iteration, then COMPARE-MEANS and SORT-MEANS compare p to the same set of means. For points that do change class, we expect COMPARE-MEANS to generally perform slightly fewer comparisons (because `minDist` $<$ `inClassDist`), and indeed this is true for all the instances described below. The overhead of COMPARE-MEANS (time not spent comparing points to means) is $\Theta(k^2 d + nkd)$ which is smaller than the overhead for SORT-MEANS when $nd = o(k \log k)$, i.e. when the number of means is very large.

3 Related Work

The running time per iteration of the (unaccelerated) K-means is $O(nkd)$ per iteration. The present paper gives a method for reducing the dependence of the running time on k. The previous work on accelerating K-means is a sequence of papers by Moore *et al.* [8,9], which develop the following idea for reducing the dependence on n. First, build a data structure such as a k-d-tree or metric tree, which partitions the dataset and essentially gives a pre-classification of the data. Then, for each iteration of K-means, rather than running through the point set, one can traverse the tree. At each node in the tree, it may be possible to show that some classes cannot contain any of the points represented by that node. When the search reaches a node for which there is only one class that provably contains all the points represented by the node, the points can be assigned to that class as a group. The assignment is fast as long as "cached sufficient statistics" (such as the mean of the represented points) are kept at each node.

Section 4.3 analyzes a dataset used by Moore [8], and finds that SORT-MEANS performs about the same number of comparisons. However, the acceleration described in this paper has two advantages: its extreme simplicity and low overhead. Regarding simplicity, the loop of Step 1b of SORT-MEANS differs from the innermost loop of standard K-means by having just two extra array accesses and a conditional termination of the loop, while COMPARE-MEANS is simpler still. The acceleration can be very easily added to existing or future implementations of K-means. The low overhead is also important, especially on lower-dimensional data — on the Landsat data described below, SORT-MEANS performs more comparisons than COMPARE-MEANS, but runs much faster because little time is spent outside of comparisons.

Because the approach of Moore *et al.* reduces dependence on n, while our approach reduces dependence on k, it is an interesting open question whether the two approaches can be combined to achieve further acceleration.

While this paper considers only K-means and related algorithms, there is a large body of work on other algorithms for clustering multi-dimensional data; see for example [2,3,4].

4 Experimental Results

The algorithms were run on a collection of datasets derived from a Landsat satellite image, a forestry application, and an Internet application. The behaviour of COMPARE-MEANS and SORT-MEANS are investigated in detail on the Landsat image, while the other datasets are used to verify the effectiveness of the acceleration in very different situations.

4.1 Landsat Dataset

The primary dataset used here is drawn from a Landsat 7 image of the north of Vietnam, specifically the image in path 128, row 44, taken on December 27 1999. The image has three visible and three near-infrared to mid-infrared bands. Each band is a 7051 $\times$ 7911 matrix of brightness values, each a number in the range 0 to 255. A sample of 2,370,686 pixels was drawn by selecting every fourth row and every fourth column of the image, retaining only pixels with non-zeroes in all bands.

In all runs described below, the influence of using Isodata remained limited to the first few iterations. Some classes were discarded and others split during early iterations; no discarding or splitting occurs at all for $k < 330$ (so the behaviour is identical to K-means), and for $k < 400$, all discarding and splitting occurs in the first two iterations. The Isodata parameters determining this behaviour were as follows: `minimum_class_size` 10 points, `maximum_standard_deviation` 10.0, `minimum_distance` 0 (so no classes get merged).

The accelerated algorithms were run with k ranging from 10 to 500 in increments of 10, and the standard algorithm was run with 10, 50, 100, 200, 300, 400 and 500 means. All runs were terminated after 100 iterations, although none had converged (i.e. reached an iteration in which no point changed class).

Figure 1 investigates the quantity γ, the number of comparisons per point made by SORT-MEANS (the number of times it computes the distance between a point and a mean) in each iteration. Considering the line for $k = 100$, we see that in the first iteration, a typical point is compared against 82 means, out of a maximum of 100. However, the number of comparisons per point drops precipitously, reaches 6 in the 5th iteration, and is below 5.1 by the 15th iteration. It quickly levels out, dropping only to 4.95 by the 100th iteration. For comparison, the number of comparisons per point without acceleration is exactly k. Thus for all but the first few iterations in the run with $k = 100$, acceleration reduces the number of comparisons by a factor of 20. COMPARE-MEANS performs almost exactly as many comparisons as SORT-MEANS on this data.

Figure 2 shows that the reduction in the number of comparisons causes a corresponding reduction in the total running time (note the log scale). For 100

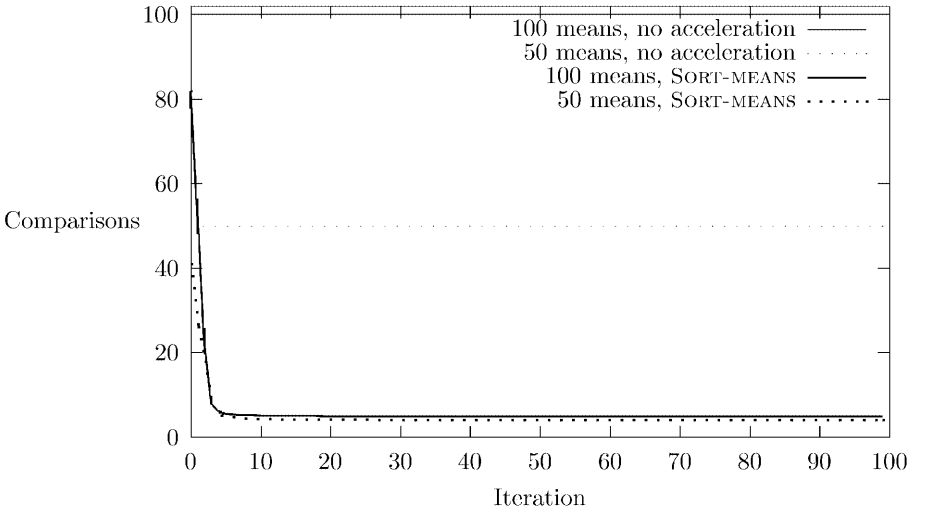

Fig. 1. Average number of comparisons per point (γ) in each iteration

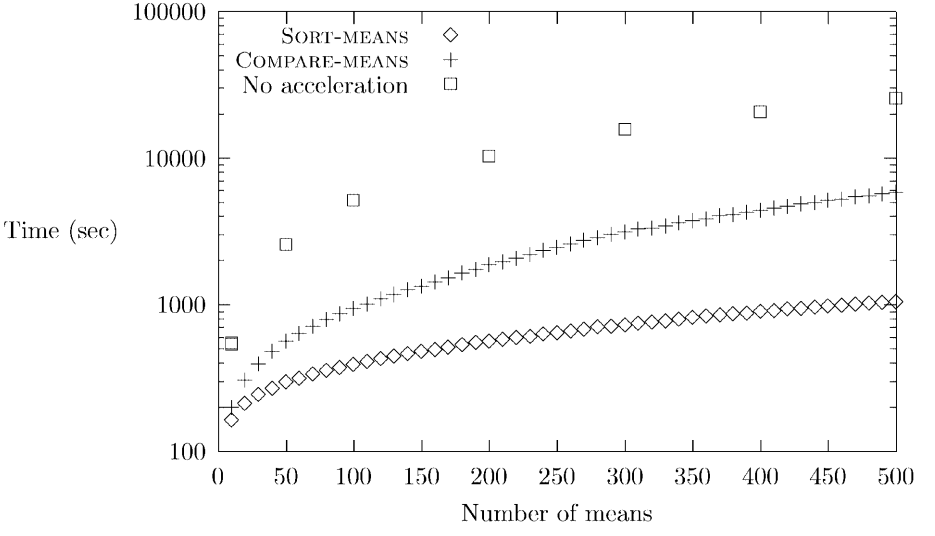

Fig. 2. Total runtime for varying numbers of means

means, the running time drops from 5115 to 945 seconds for Compare-means and 390 seconds for Sort-means, i.e. from an hour and a half to six and a half minutes. Quoted running times are for a Java implementation, running with the Java 1.3 runtime on a 850 MHz Pentium III running Windows 98. The same code was run on a Silicon Graphics Power Challenge with 270MHz MIPS R12000 IP27 processors, with running times consistently 4 to 4.5 times slower than those quoted here.

The overhead for comparing and sorting the means during these runs is very small. It is greatest for the run with 500 means, for which it is less than 400 milliseconds per iteration for Sort-means, or about 4% of the total running time (and much less for Compare-means).

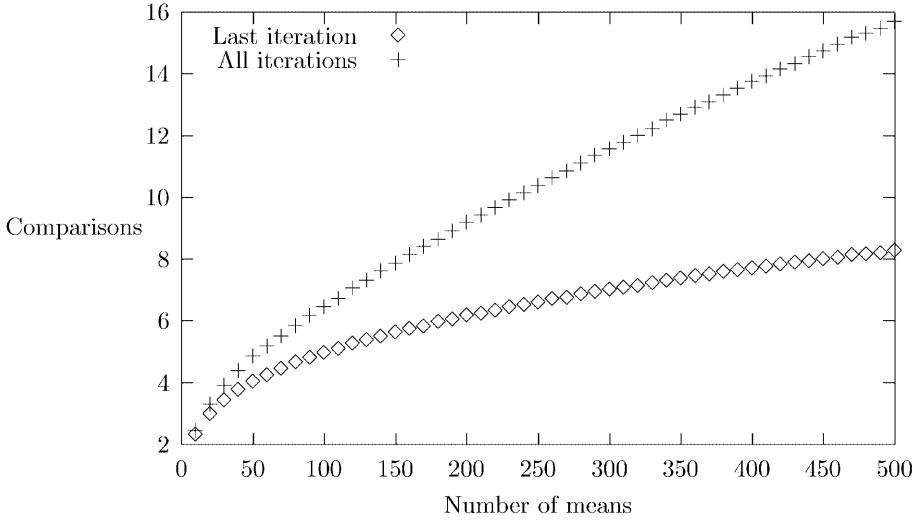

Fig. 3. Average number of comparisons per point averaged over all iterations, and in last iteration. Algorithm Sort-means.

In Figure 3 we investigate how the average number of comparisons per point varies with the number of means for Sort-means (Compare-means is almost identical for this dataset). The upper line shows the number of comparisons per point averaged over all 100 iterations. This number is strongly influenced by the comparisons made in the first few iterations. The lower line shows only the number of comparisons per point in the last iteration. The latter value is of interest if we wish to run more than 100 iterations, and therefore get a better clustering of the points, as the number of comparisons per iteration is nearly monotonically decreasing. For this dataset, the number of comparisons per point in the last iteration is less than $\log_2 k$.

The difference between the upper and lower lines in Figure 3 can be reduced by using a better initial set of means, and by making a good initial assignment of points to classes. For example, consider the following pre-classification process for a 6-dimensional dataset. First, run 20 iterations of Isodata on a subsample of a tenth of the points; this gives a good initial set of means. Then generate an initial assignment of points to classes as follows: create a six-dimensional array *initial*, where $initial[i_0][i_1][i_2][i_3][i_4][i_5]$ is the closest mean to a hypothetic point whose value in dimension j is $-1.25+i_j/2$ standard deviations from the mean, for $i_j = 0 \ldots 5$. For each dimension j, generate an array dev_j that gives the number of standard deviations from the mean for each of the 256 possible pixel values. Then the initial assignment of each point is simply determined by indexing into the *dev* and *initial* arrays. For $k = 400$, this procedure reduces the time for the first iteration of Sort-means on the large point set from 195 seconds to 36.7, the second iteration from 147 seconds to 5.2 and the total running time from 896 seconds to 581 seconds (including pre-classification). Because of the better initial set of means, the distortion at the end of 100 iterations on the large point set is smaller than without pre-classification. In fact, after 239 seconds and 30 iterations on the large set, the distortion is lower than after 100 iterations without pre-classification.

We can simplify this bootstrapping process, with only a small time penalty, by not calculating the *dev* and *initial* arrays, instead using Compare-means for the first iteration on the large point set to take advantage of the good initial means. The time for the first iteration is then 52.2 seconds, giving a total running time of 597 seconds.

Other methods could be used to obtain a good initial clustering, for example the algorithm of Gonzalez [5], which computes a 2-approximation for the problem of minimizing the maximum intra-cluster distance.

4.2 More Landsat Datasets

First the Landsat 7 image was sampled at various frequencies, producing varying numbers of points. For example, for the smallest dataset, every eighth row and column of the original image was retained. For each sampling frequency, 3 dimensional data corresponding only to the visible bands was considered, in addition to the 6 dimensional data that includes the visible, near and mid-infrared bands. Each run used $k = 100$ and 100 iterations. The results for Sort-means are shown in Table 1, where "avg comparisons" refers to the number of comparisons per point averaged over all iterations, and "end comparisons" refers to the last iteration. We see that the absolute speedup ranges from a factor of 13 to a factor of 16, consistent with the primary dataset above.

4.3 Forest Cover Type

The next dataset is the Forest Cover Type dataset from the UCI Knowledge Discovery in Databases Archive [1]. This dataset, which was also used as a test instance by Moore [8], relates forest cover type for 30 x 30 meter cells (obtained

Table 1. Performance on various samples of the Landsat 7 image.

		Comparisons		Time (seconds)		
Points	Bands	Avg	End	SORT-MEANS	No acceleration	Speedup
2408250	3	4.765	3.581	176	2872	16
1053635	6	6.428	4.951	182	2316	13
1070341	3	4.753	3.565	79	1279	16
592674	6	6.437	4.947	99	1275	13
602070	3	4.745	3.578	48	720	15

from US Forest Service Region 2 Resource Information System data) to various variables, such as distance from roads, elevation and soil type. There are 581012 data points in 54 dimensions (though Moore gives the number of data points as 150000). Ten of the dimensions contain integer quantitative data, while the remaining 44 are binary, though all 54 are handled identically by K-means.

Table 2. Performance on the Forest Cover Type dataset.

	Comparisons (avg)	Comparisons (last iteration)	Time
No acceleration	100.0	100.0	3:09:00
COMPARE-MEANS	5.488	5.253	14:47
SORT-MEANS	6.149	5.297	13:00

Table 2 shows the result of running 100 iterations with $k = 100$, and initial means randomly selected from the set of points. We see that Algorithm COMPARE-MEANS reduces the number of comparisons by a factor of 18, and reduces the time by a factor of 13. Algorithm SORT-MEANS gives a slight improvement in speed, giving a factor of 15 speedup over ordinary K-means.

The number of comparisons made by both algorithms compares is slightly higher than is reported in Moore [8], where a reduction of the number of comparisons by a factor of 19 is reported. However, it is difficult to make a direct comparison, as the reported number of data points is different. In addition, run time is not reported in [8], and it is not clear whether the k-d-tree operations constitute a significant fraction of run time.

4.4 BGP Updates

The last dataset involves the Border Gateway Protocol (BGP) running on the global Internet. All BGP updates received from the global Internet by AT&T were observed for 29 days. For each IP address block (or routing table entry), the log of the number of updates for each of the 29 days was recorded. The resulting real-valued data consists of 140980 "points" in 29 dimensions.

A cluster of similar points corresponds to a set of IP addresses all generating similar numbers of BGP updates on the same days, perhaps all affected by a single fault in the network. Clustering the dataset may therefore help to identify the source and extent of network faults. The logarithm is used because it better models the intuitive notion of a similar number of updates.

This dataset has a small natural cluster size, so the clustering algorithms were run with a large number of means (5000). The initial means were randomly chosen, and the runs converged after 27 iterations. For this data set, SORT-MEANS was run with a first iteration of COMPARE-MEANS, taking advantage of the widely separated initial means (see the end of Section 2.2 for more discussion). The performance of the algorithms is shown in Figure 3.

Table 3. Performance on the BGP updates dataset.

	Comparisons (avg)	Comparison / sort time	Total time
No acceleration	5000.0	0	15:51:16
COMPARE-MEANS	365.153	21:57	1:24:08
SORT-MEANS	366.933	42:54	1:34:55

Algorithm COMPARE-MEANS achieves a speedup of 11.3, and performs a factor of 13.6 fewer comparisons than standard Isodata. Unlike the previous datasets, the algorithm SORT-MEANS is slower than COMPARE-MEANS, even with a first iteration using COMPARE-MEANS, because of the overhead of sorting the large number of means. Indeed, algorithm SORT-MEANS spends almost half of its time sorting the means.

5 Conclusions and Extensions

This paper has presented two modifications for accelerating K-means: avoiding making unnecessary comparisons between data points and means by comparing means to each other (algorithm COMPARE-MEANS), and avoiding overhead by sorting the means (algorithm SORT-MEANS). The modifications offer very significant speedups for K-means and related classification algorithms on a wide range of datasets. The modifications are simple enough to be easily added to existing implementations of clustering algorithms.

Algorithm SORT-MEANS is significantly faster than algorithm COMPARE-MEANS (except when the number of means is very large), and the difference is most pronounced when the number of dimensions of the dataset is small.

Acknowledgements. Thanks to the Center for Biodiversity and Conservation at the American Museum of Natural History for the opportunity to learn about remote sensing and for the use of the Landsat 7 image, to Ned Horning at Spatial

Support Services for conversations on remote sensing, and to Sanjoy Dasgupta and Howard Karloff at AT&T Labs-Research for conversations on the K-means algorithm and clustering. Thanks to Carsten Lund and Nick Reingold at AT&T Labs-Research for providing the BGP-update dataset.

References

1. S. D. Bay. The UCI KDD Archive [http://kdd.ics.uci.edu]. Irvine, CA: University of California, Department of Information and Computer Science., 1999.
2. P. S. Bradley, U. M. Fayyad, and C. Reina. Scaling clustering algorithms to large databases. In *Knowledge Discovery and Data Mining*, pages 9–15, 1998.
3. A. Dempster, N. Laird, and D. Rubin. Maximum-likelihood from incomplete data via the EM algorithm. *J. Royal Statistical Society B*, 39:1–38, 1977.
4. F. Farnstrom, J. Lewis, and C. Elkan. Scalability for clustering algorithms revisited. *SIGKDD Explorations*, 2(1):51–57, 2000.
5. T. F. Gonzalez. Clustering to minimize the maximum intercluster distance. *Theoretical Computer Science*, 38:293–306, 1985.
6. J. R. Jensen. *Introductory Digital Image Processing, A Remote Sensing Perspective.* Prentice Hall, Upper Saddle River, NJ, 1996.
7. J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proc. Fifth Berkeley Symposium on Mathematics, Statistics and Probability*, volume 1, pages 281–296, 1967.
8. A. W. Moore. The anchors hierarchy: Using the triangle inequality to survive high dimensional data. In *Proc. UAI-2000: The Sixteenth Conference on Uncertainty in Artificial Intelligence*, 2000.
9. D. Pelleg and A. W. Moore. Accelerating exact k-means algorithms with geometric reasoning. In *Proc. Fifth International Conference on Knowledge Discovery and Data Mining.* AAAI Press, 1999.
10. J. T. Tou and R. C. Gonzalez. *Pattern Recognition Principles.* Addison-Wesley, Reading, MA, 1977.