Alternatives to the k-means algorithm that find better clusterings

Greg Hamerly GHAMERLY@CS.UCSD.EDU
Charles Elkan ELKAN@CS.UCSD.EDU
Department of Computer Science and Engineering, University of California, San Diego, La Jolla, CA 92093

Abstract

We investigate here the behavior of the standard k-means clustering algorithm and several alternatives to it: the k-harmonic means algorithm due to Zhang and colleagues, fuzzy k-means, Gaussian expectation-maximization, and two new variants of k-harmonic means. Our aim is to find which aspects of these algorithms contribute to finding good clusterings, as opposed to converging to a low-quality local optimum. We describe each algorithm in a unified framework that introduces separate cluster membership and data weight functions. We then show that the algorithms do behave very differently from each other on simple low-dimensional synthetic datasets, and that the k-harmonic means method is superior. Having a soft membership function is essential for finding high-quality clusterings, but having a non-constant data weight function is useful also.

1. Introduction

Data clustering, which is the task of finding natural groupings in data, is an important task in machine learning and pattern recognition. Typically in clustering there is no one perfect solution to the problem, but algorithms seek to minimize a certain mathematical criterion (which varies between algorithms). Minimizing such criteria is known to be NP-hard for the general problem of partitioning d-dimensional data into k sets (Ostrovsky & Rabani, 2000; Drineas et al., 1999). Algorithms like k-means seek local rather than the global minimum solutions, but can get stuck at poor solutions. In these cases we consider that a solution which better minimizes the mathematical criterion (for the same number of centers) to be a better-quality clustering.

We use the term "center-based clustering" to refer to the family of algorithms such as k-means and Gaussian expectation-maximization, since they use a number of "centers" to represent and/or partition the input data. Each center defines a cluster with a central point and perhaps a covariance matrix. Center-based clustering algorithms begin with a guess about the solution, and then refine the positions of centers until a local optimum is reached. These methods can work well, but they can also converge to a local minimum that is far from the global minimum, i.e. the clustering that has the highest quality according to the criterion in use. Converging to bad local optima is related to sensitivity to initialization, and is a primary problem of data clustering.

The goal of this work is to understand and extend center-based clustering algorithms to find good-quality clusterings in spatial data. Recently, many wrapper methods have been proposed to improve clustering solutions. A wrapper method is one that transforms the input or output of the clustering algorithm, and/or uses the algorithm multiple times. One wrapper method is simply running the clustering algorithm several times from different starting points (often called random restart), and taking the best solution (Jain et al., 1999). This is a common technique, and methods such as used in (Likas et al., 2001) push this technique to its extreme, at the cost of computation. Another wrapper method to finding quality clusterings is using the best initializations possible; this has been looked at in (Peña et al., 1999; Meĭla & Heckerman, 2001; Bradley & Fayyad, 1998). This is fruitful research, as many clustering algorithms are sensitive to their initializations. Other research (Sand & Moore, 2001; Pelleg & Moore, 2000) has been looking at finding the appropriate number of clusters, and analyzing the difference between the cluster solution and the dataset. This is useful when the appropriate number of centers is unknown, or the algorithm is stuck at a sub-optimal solution.

These approaches are beneficial, but they are attempting to fix the problems of clustering algorithms externally, rather than to improve the clustering algorithms themselves. We are interested in improving the clustering algorithms directly to make them less sensitive to initializations and give better solutions. Of course, any clustering algorithm developed could benefit from wrapper methods.

Recently, Zhang *et al.* introduced a new clustering algorithm called k-harmonic means (KHM) that arises from an optimization criterion based on the harmonic mean (Zhang et al., 1999; Zhang, 2000). This algorithm shows promise in finding good clustering solutions quickly, and outperforms k-means (KM) and Gaussian expectation-maximization (GEM) in many tests. The KHM algorithm also has a novel feature that gives more influence to data points that are not well-modeled by the clustering solution, but is unknown how important this feature is. Our work is a first answer to this question.

In this paper, we present a unified framework for looking at center-based clustering algorithms, and then derive two new algorithms that are based on properties of KM and KHM. The algorithms are compared analytically and empirically.

2. Center-based clustering algorithms

The algorithms k-means, Gaussian expectation-maximization, fuzzy k-means, and k-harmonic means are in the family of center-based clustering algorithms. They each have their own objective function, which defines how good a clustering solution is. The goal of each algorithm is to minimize its objective function. Since these objective functions cannot be minimized directly, we use iterative update algorithms which converge on local minima.

2.1 General iterative clustering

We can formulate a general form for the family of clustering algorithms that use iterative optimization (Kalton et al., 2001), and use this framework to make comparisons between algorithms. Define a d-dimensional set of n data points $X = \{x_1, \ldots, x_n\}$ as the data to be clustered. Define a d-dimensional set of k centers $C = \{c_1, \ldots, c_k\}$ as the clustering solution that an iterative algorithm refines.

A membership function $m(c_j|x_i)$ defines the proportion of data point x_i that belongs to center c_j with constraints $m(c_j|x_i) \geq 0$ and $\sum_{j=1}^{k} m(c_j|x_i) = 1$. Some algorithms use a *hard* membership function, meaning $m(c_j|x_i) \in \{0, 1\}$, while others use a *soft* membership function, meaning $0 \leq m(c_j|x_i) \leq 1$. Kearns and colleagues have analyzed the differences between hard and soft membership from an information-theoretic standpoint (Kearns et al., 1997). One of the reasons that k-means can converge to poor solutions is due to its hard membership function. However, the hard membership function makes possible many computational optimizations that do not affect accuracy of the algorithm, such as using kd-trees (Pelleg & Moore, 1999).

A weight function $w(x_i)$ defines how much influence data point x_i has in recomputing the center parameters in the next iteration, with constraint $w(x_i) > 0$. This weight function was introduced in (Zhang, 2000). Giving variable influence to data in clustering has analogies to boosting in supervised learning (Freund & Schapire, 1999). Each approach gives more weight to data points that are not "well-covered" by the current solution. However, unlike boosting in supervised learning, this approach does not create an ensemble of solutions.

Now we can define a general form of iterative, center-based clustering. The steps are:

1. Initialize the algorithm with guessed centers C.
2. For each data point x_i, compute its membership $m(c_j|x_i)$ in each center c_j and its weight $w(x_i)$.
3. For each center c_j, recompute its location from all data points x_i according to their memberships and weights:
$$c_j = \frac{\sum_{i=1}^{n} m(c_j|x_i) w(x_i) x_i}{\sum_{i=1}^{n} m(c_j|x_i) w(x_i)} \quad (1)$$
4. Repeat steps 2 and 3 until convergence.

Now we can compare algorithms based on their membership and weight functions. An alternative initialization procedure is to guess an initial partition, and then start the algorithm from step 3. The computational complexity of each algorithm in this paper is $O(nkd)$ for each update iteration (Equation 1). The algorithms vary by constant factors but have the same order complexity.

2.2 The k-means algorithm

The k-means algorithm (KM) (MacQueen, 1967) partitions data into k sets. The solution is then a set of k centers, each of which is located at the centroid of the data for which it is the closest center. For the membership function, each data point belongs to its nearest center, forming a Voronoi partition of the data. The objective function that the KM algorithm optimizes is

$$KM(X, C) = \sum_{i=1}^{n} \min_{j \in \{1 \ldots k\}} ||x_i - c_j||^2 \quad (2)$$

This objective function gives an algorithm which minimizes the within-cluster variance (the squared distance between each center and its assigned data points).

The membership and weight functions for KM are:

$$m_{KM}(c_l|x_i) = \begin{cases} 1 \; ; \; \text{if } l = \arg\min_j ||x_i - c_j||^2 \\ 0 \; ; \; \text{otherwise} \end{cases} \tag{3}$$

$$w_{KM}(x_i) = 1 \tag{4}$$

KM has a hard membership function, and a constant weight function that gives all data points equal importance. KM is easy to understand and implement, making it a popular algorithm for clustering.

2.3 The Gaussian expectation-maximization algorithm

The Gaussian expectation-maximization (GEM) algorithm for clustering uses a linear combination of d-dimensional Gaussian distributions as the centers. It minimizes the objective function

$$GEM(X, C) = -\sum_{i=1}^{n} \log \left(\sum_{j=1}^{k} p(x_i|c_j)p(c_j) \right) \tag{5}$$

where $p(x_i|c_j)$ is the probability of x_i given that it is generated by the Gaussian distribution with center c_j, and $p(c_j)$ is the prior probability of center c_j. We use a logarithm to make the math easier (while not changing the solution), and we negate the value so that we can minimize the quantity (as we do with the other algorithms we investigate). See (Bishop, 1995, pages 59–73) for more about this algorithm. The membership and weight functions of GEM are

$$m_{GEM}(c_j|x_i) = \frac{p(x_i|c_j)p(c_j)}{p(x_i)} \tag{6}$$

$$w_{GEM}(x_i) = 1 \tag{7}$$

Bayes' rule is used to compute the soft membership, and m_{GEM} is a probability since the factors in Equation 6 are probabilities. GEM has a constant weight function that gives all data points equally importance, like KM. Note that $w_{GEM}(x_i)$ is not the same as $p(x_i)$.

2.4 The fuzzy k-means algorithm

The fuzzy k-means algorithm (FKM; also called fuzzy c-means) (Bezdek, 1981) is an adaptation of the KM algorithm that uses a soft membership function. Unlike KM which assigns each data point to its closest center, the FKM algorithm allows a data point to belong partly to all centers, like GEM.

$$FKM(X, C) = \sum_{i=1}^{n} \sum_{j=1}^{k} u_{ij}^r ||x_i - c_j||^2 \tag{8}$$

The parameter u_{ij} denotes the proportion of data point x_i that is assigned to center c_j, and is under the constraints $\sum_{j=1}^{k} u_{ij} = 1$ for all i and $u_{ij} \geq 0$. The parameter r has the constraint $r \geq 1$. A larger value for r makes the method "more fuzzy."

Bezdek and others give separate update functions for u_{ij} and c_j. The u_{ij} update equation depends only on C and X, so we incorporate its update function into the update for c_j. Then we can represent FKM in the form of the general iterative update of Equation 1. The membership and weight functions for FKM are:

$$m_{FKM}(c_j|x_i) = \frac{||x_i - c_j||^{-2/(r-1)}}{\sum_{j=1}^{k} ||x_i - c_j||^{-2/(r-1)}} \tag{9}$$

$$w_{FKM}(x_i) = 1 \tag{10}$$

so FKM has a soft membership function, and a constant weight function. As r tends toward 1 from above, the algorithm behaves more like standard k-means, and the centers share the data points less.

2.5 k-harmonic means

The k-harmonic means algorithm (KHM) is a method similar to KM that arises from a different objective function (Zhang, 2000). The KHM objective function uses the harmonic mean of the distance from each data point to all centers.

$$KHM(X, C) = \sum_{i=1}^{n} \frac{k}{\sum_{j=1}^{k} \frac{1}{||x_i - c_j||^p}} \tag{11}$$

Here p is an input parameter, and typically $p \geq 2$. The harmonic mean gives a good (low) score for each data point when that data point is close to any one center. This is a property of the harmonic mean; it is similar to the minimum function used by KM, but it is a smooth differentiable function.

The membership and weight functions for KHM are:

$$m_{KHM}(c_j|x_i) = \frac{||x_i - c_j||^{-p-2}}{\sum_{j=1}^{k} ||x_i - c_j||^{-p-2}} \tag{12}$$

$$w_{KHM}(x_i) = \frac{\sum_{j=1}^{k} ||x_i - c_j||^{-p-2}}{\left(\sum_{j=1}^{k} ||x_i - c_j||^{-p} \right)^2} \tag{13}$$

Note that KHM has a soft membership function, and also a varying weight function. This weight function gives higher weight to points that are far away from every center, which aids the centers in spreading to cover the data.

The implementation of KHM needs to deal with the case where $x_i = c_j$. In this case we follow Zhang using $\max(||x_i - c_j||, \epsilon)$ and use a small positive value of ϵ. We also apply this technique for FKM and the algorithms discussed in Section 3. We have not encountered any numerical problems in any of our tests.

3. New clustering algorithms

We are interested in the properties of the new algorithm KHM. It has a soft membership function and a varying weight function, which makes it unique among the algorithms we have encountered. KHM has been shown to be less sensitive to initialization on synthetic data (Zhang, 2000).

Here we analyze two aspects of KHM (the membership and the weight functions) and define two new algorithms we call Hybrid 1 and Hybrid 2. They are named for the fact that they are hybrid algorithms that combine features of KM and KHM. The purpose for creating these algorithms is to find out what effects the membership and weight functions of KHM have by themselves.

3.1 Hybrid 1: hard membership, varying weights

Hybrid 1 (H1) uses the hard membership function of KM. Every point belongs only to its closest center. What makes H1 different from KM is the KHM weight function, which gives more weight to points that are far from every center. We expect that this algorithm should converge more quickly than KM due to the weights, but will still have problems related to the hard membership function. As far as we know, adding weights in this manner to KM is a new idea.

The definitions of the membership and weight functions for H1 are:

$$m_{H1}(c_l|x_i) = \begin{cases} 1 \; ; \; \text{if } l = \arg\min_j ||x_i - c_j||^2 \\ 0 \; ; \; \text{otherwise} \end{cases} \qquad (14)$$

$$w_{H1}(x_i) = \frac{\sum_{j=1}^{k} ||x_i - c_j||^{-p-2}}{\left(\sum_{j=1}^{k} ||x_i - c_j||^{-p}\right)^2} \qquad (15)$$

3.2 Hybrid 2: soft membership, constant weights

Hybrid 2 (H2) uses the soft membership function of KHM, and the constant weight function of KM. The definitions of the membership and weight functions for H2 are:

$$m_{H2}(c_j|x_i) = \frac{||x_i - c_j||^{-p-2}}{\sum_{j=1}^{k} ||x_i - c_j||^{-p-2}} \qquad (16)$$

$$w_{H2}(x_i) = 1 \qquad (17)$$

Note that H2 resembles FKM. In fact, for certain values of r and p they are mathematically equivalent. It is interesting to note, then, that the membership function of KHM (from which we get H2) and FKM are also very similar. We investigate H2 and FKM as separate entities to keep clear the fact that we are investigating the membership and weight functions of KHM separately.

4. Experimental setup

We perform two sets of experiments to demonstrate the properties of the algorithms described in Sections 2 and 3. We want to answer several questions: how do different initializations affect each algorithm, what is the influence of soft versus hard membership, and what is the benefit of using varying versus constant weights.

Though each algorithm minimizes a different objective function, we measure the quality of each clustering solution by the square-root of the k-means objective function in Equation 2. It is a reasonable metric by which to judge cluster quality, and by using a single metric we can compare different algorithms. We use the square root because of the squared distance term in the function which can exaggerate the severity of poor solutions. We considered running KM on the output of each algorithm, so that the KM objective function could be better minimized. We found that this did not help significantly, so we do not do this here.

Our experiments use two datasets already used in recent empirical work on clustering algorithms (Zhang et al., 1997; Pelleg & Moore, 1999). The algorithms we test are KM, KHM, FKM, H1, H2, and GEM. The code for each of these algorithms is our own, except for GEM. Our code is written in Matlab; for GEM we used the FastMix implementation provided by (Sand & Moore, 2002). We need to supply the the KHM, H1, and H2 with the parameter p, and FKM with r. We set $p = 3.5$ for all tests, as that was the best value found by Zhang. We set $r = 1.3$, as that is the best value we found based on our preliminary tests.

The two initializations we use are the Forgy and Random Partition methods (Peña et al., 1999). The Forgy method simply chooses k data points from the dataset at random and uses them as the initial centers. The Random Partition method assigns each data point to a random center, then computes the initial location of each center as the centroid of its assigned points. The Forgy method tends to spread centers out in the data, while the Random Partition method tends to place the centers in a small area near the middle of the dataset.

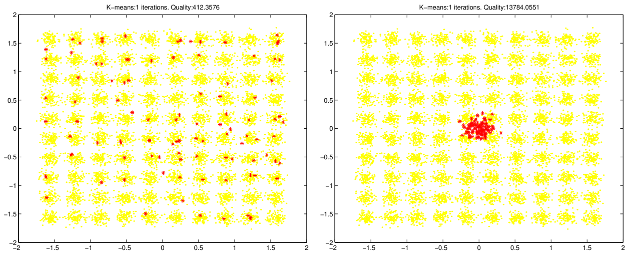

Figure 1. Experiment 1: Forgy (left) and Random Partition (right) initializations for the BIRCH dataset. Centers are shown in the dark color, data points in the light color. This dataset has a grid of 10x10 natural clusters.

Random Partition was found to be a preferable initialization method for its simplicity and quality in (Peña et al., 1999; Meĭla & Heckerman, 2001). For GEM, we also initialize $p(c_j) = 1/k$ and initialize the covariance to be $0.2I$, where I is the identity matrix.

Before clustering, all datasets used in both experiments are shifted and re-scaled to give each dimension zero mean and unit variance. This is the standard z-score transformation. This can be a good idea before using algorithms based on distance metrics (as these are), as it allows each dimension to have the same influence. Without such scaling, computed distances may be more influenced by some dimensions.

4.1 Experiment 1: BIRCH

The purpose of our first experiment is to illustrate the convergence properties of the different algorithms, and to show the need to improve clustering algorithms. We use a randomly generated synthetic dataset we call BIRCH, as defined by (Zhang et al., 1997). This dataset has $k = 100$ true clusters arranged in a 10x10 grid in $d = 2$ dimensions. Each cluster generates 100 data points from its own Gaussian distribution, for a total of $n = 10,000$ data points. The distance between two adjacent cluster means is $4\sqrt{2}$ with cluster radius of $\sqrt{2}$ (meaning the variance in each dimension is 1). We run each algorithm twice, once with the Forgy initialization, and once with the Random Partition initialization. Figure 1 shows the two initializations. We use the same randomly chosen initializations for all algorithms. All our results are similar for other random initializations.

4.2 Experiment 2: Pelleg and Moore data

The second experiment uses a synthetic dataset based on work by (Pelleg & Moore, 1999). Here we run many tests to determine the average-case behavior of the algorithms. We test datasets of dimensions 2, 4, and 6 to show that all these algorithms work well in low dimensions. Each dataset has $k = 50$ true natural clusters which generate $n = 2500$ total data points. The true cluster centers are chosen at random in the unit hypercube, then 2500 data points are generated by choosing a cluster randomly, and generating a data point according to a Gaussian distribution with standard deviation $s = d \times 0.012$ and mean at the true cluster center. We generate data that is more separated (clusters have less overlap) than the work by Pelleg and Moore (who used $s = d \times 0.025$), because this presents a more difficult task to the clustering algorithms. This is because it is harder for centers to move freely through the whole dataset due to gaps between natural clusters. See figure 2 for a simple example.

For each $d \in \{2, 4, 6\}$ we generate 100 datasets, and two initializations (Random Partition and Forgy) for each dataset. Then we test each algorithm from both of these initializations. For each algorithm we allow it to run for 100 iterations, which is plenty for the algorithms to converge.

4.3 Dimensionality reduction

Clustering in high dimensions is difficult. Most clustering algorithms that are based on distance metrics fail to find good clusterings in high dimensions, because the distance to all data points increases with increasing dimension, making it harder to discriminate between points of different clusters when a center is far away.

Some clustering methods are designed for high dimensional data, such as text clustering described in (Zhao & Karypis, 2001). This method uses the cosine distance (normalized dot product) to determine the distance between data points. However, methods based on this metric fail to use the magnitude of the data points, making it impossible to discriminate between clusters that lie along the same vector from the origin.

Recent research indicates that it may be preferable to use dimensionality reduction techniques prior to clustering, rather than clustering directly in high dimensions. In (Dasgupta, 2000) the author shows that highly eccentric clusters in high dimensions tend to become more spherical in lower dimensions while remaining well-separated, even when the dimension reduction done is a random linear projection. This fact makes the clustering task even easier, and at the cost of a very inexpensive operation (linear projection).

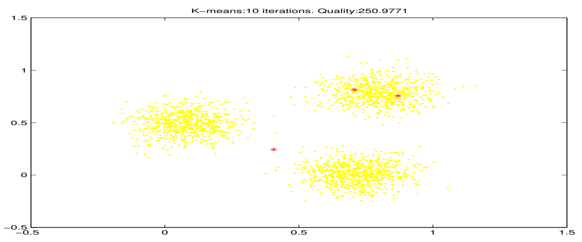

Figure 2. Here k-means has converged to a local optimum which it would be able to escape if the clusters had more overlap. Without overlap, one of the "trapped" centers on the top-right cannot move to a cluster on the bottom or left because of the hard assignment function. Thus, a clustering problem is harder when clusters have less overlap, because the probability of finding bad local optima is higher.

Table 1. Experiment 1: Quality of solutions for one run on the BIRCH dataset, using Forgy and Random Partition initializations. Lower quality scores are better. "Clusters found" is the number of true clusters (maximum 100) in which the algorithm placed at least one center.

	$\sqrt{KM}$ **quality**		**Clusters found**	
	Forgy	RP	Forgy	RP
GEM	15.530	24.399	77	49
KM	12.771	18.396	83	60
H1	12.159	15.242	86	72
FKM	11.612	10.441	89	93
H2	10.670	9.908	92	95
KHM	10.255	9.999	94	95

5. Experimental results

5.1 Experiment 1: BIRCH

Running each algorithm on the BIRCH dataset once gives an intuition for how each behaves. Figure 1 shows the two initializations we use. Then the end results of each algorithm's run is shown in Figure 3 and the cluster qualities are shown in Table 1. FKM, H2, and KHM all found good clusterings for both types of initializations, and they are all soft membership algorithms.

We can see that the two hard membership algorithms, KM and H1, have distinctly different behavior for the two initializations. The Forgy initialization provides reasonably good behavior for those two algorithms, but the Random Partition tends to leave a density of centers in the middle of the dataset that are "trapped". This is because the hard membership function prevents centers from moving if they do not own enough points, which is why hard membership can be called "winner take all". In Table 1 we show the number of true clusters found, which is the number of true clusters (out of 100) that received a center by the algorithm.

Although GEM has a soft membership function, it does poorly on this dataset due to some centers having variance that is too large and taking over several clusters. Note that the result of Random Partition for GEM appears to have more density concentrated in the middle of the dataset, where the centers began. This is similar to the hard membership results. The FastMix implementation we used for GEM started with 100 centers and would remove centers whose prior became too small. For this reason, it ended with 98 and 81 centers for the Forgy and Random Partition initializations, respectively. One feature of FastMix is its ability to determine the number of centers through density estimation. We tried starting FastMix without a pre-defined number of centers, and it found 23.

5.2 Experiment 2: Pelleg and Moore data

Our second experiment shows the average performance of the algorithms compared over many randomly generated data sets in several dimensions. For each dataset $X_{d,i}$ where $d \in \{2,4,6\}$ and $1 \leq i \leq 100$ we compute the optimal KM partition $O_{d,i}$ by running KM to convergence starting with the centers that generated the data sets. Then we compute the score of a clustering $C_{d,i}$ as the ratio

$$R_{d,i} = \sqrt{\frac{KM(X_{d,i}, C_{d,i})}{KM(X_{d,i}, O_{d,i})}} \qquad (18)$$

Table 2 shows the mean and standard deviation of $R_{d,i}$ for each algorithm, computed using 100 datasets in 2 dimensions. Table 3 shows the point-wise comparison of each algorithm for the same experiment. It is clear from this as well that soft membership algorithms (KHM, FKM, H2) perform better than hard membership algorithms (KM, H1) in both average performance and variance. The results for 4 and 6 dimensional datasets are very similar, so we do not report them here.

Figure 4 shows the speed of convergence of each algorithm for $d = 2$ dimensions. The x-axis shows the iteration number, and the y-axis shows the average k-means quality ratio at that iteration, computed using the 100 datasets. We can see that GEM and KM are uniformly inferior to every other algorithm, and that the soft membership algorithms KHM, H2, and FKM move quickly to find good solutions. Only the final result for GEM is plotted as we cannot capture clustering progress before the FastMix software terminates.

FastMix has the ability to add and remove centers to

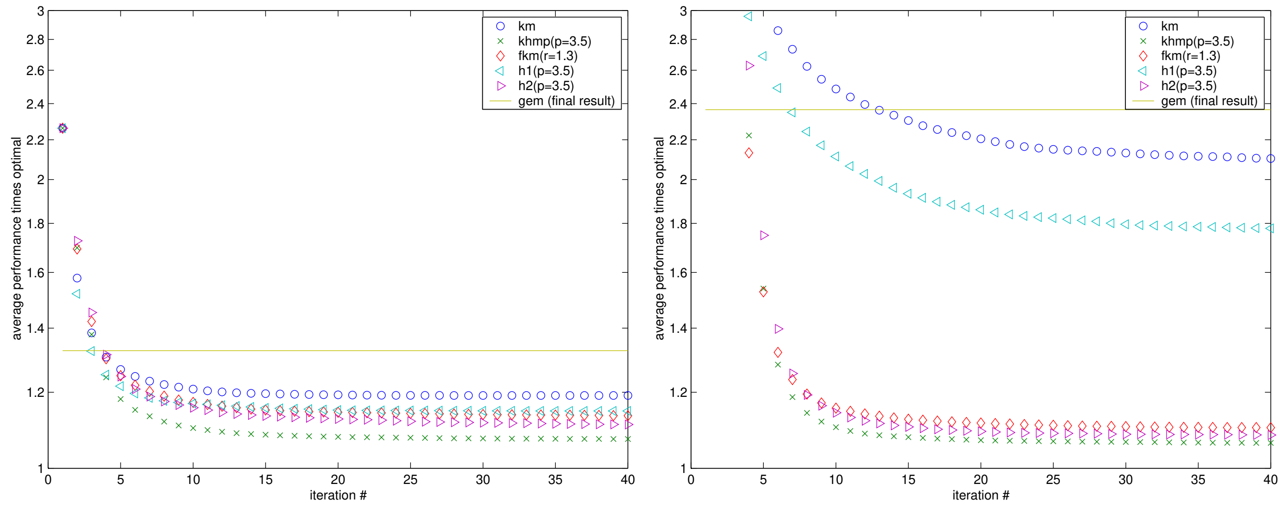

Figure 4. Experiment 2: Convergence curves starting from Forgy (top) and Random Partition (bottom) initializations on 2-d synthetic data. The x-axis shows the number of iterations, and the y-axis (log scale) shows average clustering quality score, where lower values are better. Only the final results for GEM are shown. Note that KM performs worse than every other algorithm.

better fit its data. FastMix adds a center to area where the model underpredicts the data and removes a center if its prior probability is too low. We expect that FastMix's ability to add centers would be helpful in a dataset in which clusters are well-separated. For experiment 2, FastMix began with 50 centers and only removed centers. FastMix ended with an average of 48.39 centers (Forgy) and 40.13 centers (Random Partition) in the 2-dimension test. This shows that GEM is also sensitive to poor initializations.

Table 2. Experiment 2: The mean and standard deviation of the ratio between the k-means quality and the optimum, over 100 datasets, in 2 dimensions. The ratio is computed from the square root of the k-means quality divided by the optimum k-means quality. Lower values are better. The statistics for 4 and 6 dimensions are similar, and have the same ranking.

	Forgy	Random Partition
GEM	1.3262 +/- 0.1342	2.3653 +/- 0.4497
KM	1.1909 +/- 0.0953	2.0905 +/- 0.2616
H1	1.1473 +/- 0.0650	1.7644 +/- 0.2403
FKM	1.1281 +/- 0.0637	1.0989 +/- 0.0499
H2	1.1077 +/- 0.0536	1.0788 +/- 0.0416
KHM	1.0705 +/- 0.0310	1.0605 +/- 0.0294

6. Conclusions

Our experiments clearly show the superiority of the k-harmonic means algorithm (KHM) for finding clusterings of high quality in low dimensions. Our algorithms H1 and H2 let us study the effects of the KHM weight and membership separately. They show that soft membership is essential for finding good clusterings, as H2 performs nearly as well as KHM, but that varying weights are beneficial with a hard membership function, since H1 performs better than KM. Varying weights are intuitively similar to the weights applied to training examples by boosting (Freund & Schapire, 1999). It remains to be seen whether this analogy can be made precise.

Previous work in initialization methods has concluded that the Random Partition method is good for GEM and for KM, but our experiments do not confirm this conclusion. The Forgy method of initialization (choosing random points as initial centers) works best for GEM, KM, and H1. Overall, our results suggest that the best algorithms available today are FKM, H2, and KHM, initialized by the Random Partition method. Our future work will include an investigation of performance on high-dimensional data, and on real-world datasets.

References

Bezdek, J. C. (1981). *Pattern recognition with fuzzy objective function algorithms.* New York: Plenum Press.

Bishop, C. M. (1995). *Neural networks for pattern recognition.* Clarendon Press, Oxford.

Bradley, P. S., & Fayyad, U. M. (1998). Refining initial points for K-Means clustering. *Proc. 15th International*

Table 3. Experiment 2: Competition matrix for 2-d synthetic data starting from Forgy (left) and Random Partition (right) initializations. Each entry shows the number of times (maximum 100) that the algorithm in the column had a better quality clustering than the algorithm in the row. The results for 4 and 6 dimensions are similar, so we do not report them here. In particular, KHM is better than KM for 99 to 100 of 100 times in each dimension tested.

	GEM	KM	KHM	FKM	H1	H2
GEM		97	100	100	98	100
KM	3		99	91	73	91
KHM	0	1		15	7	24
FKM	0	9	85		19	62
H1	2	27	93	81		90
H2	0	9	76	38	10	
sum:	5	143	453	325	207	367

	GEM	KM	KHM	FKM	H1	H2
GEM		100	100	100	100	100
KM	0		100	100	100	100
KHM	0	0		21	0	34
FKM	0	0	79		0	70
H1	0	0	100	100		100
H2	0	0	66	30	0	
sum:	0	100	445	351	200	404

Conf. on Machine Learning (pp. 91–99). Morgan Kaufmann, San Francisco, CA.

Dasgupta, S. (2000). Experiments with random projection. *Uncertainty in Artificial Intelligence: Proceedings of the Sixteenth Conference (UAI-2000)* (pp. 143–151). San Francisco, CA: Morgan Kaufmann Publishers.

Drineas, Frieze, Kannan, Vempala, & Vinay (1999). Clustering in large graphs and matrices. *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms).*

Freund, Y., & Schapire, R. (1999). A short introduction to boosting. *Japanese Society for Artificial Intelligence*, *14*, 771–780.

Jain, A. K., Murty, M. N., & Flynn, P. J. (1999). Data clustering: a review. *ACM Computing Surveys*, *31*, 264–323.

Kalton, A., Langley, P., Wagstaff, K., & Yoo, J. (2001). Generalized clustering, supervised learning, and data assignment. *Proceedings of the Seventh International Conference on Knowledge Discovery and Data Mining* (pp. 299–304). San Francisco, CA: ACM Press.

Kearns, M., Mansour, Y., & Ng, A. Y. (1997). An information-theoretic analysis of hard and soft assignment methods for clustering. *Proceedings of Uncertainty in Artificial Intelligence* (pp. 282–293). AAAI.

Likas, A., Vlassis, N., & Verbeek, J. (2001). *The global k-means clustering algorithm* (Technical Report). Computer Science Institute, University of Amsterdam, The Netherlands. IAS-UVA-01-02.

MacQueen, J. (1967). Some methods for classification and analysis of multivariate observations. *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability* (pp. 281–297). Berkeley, CA: University of California Press.

Meĭla, M., & Heckerman, D. (2001). An experimental comparison of model-based clustering methods. *Machine learning*, *42*, 9–29.

Ostrovsky, R., & Rabani, Y. (2000). Polynomial time approximation schemes for geometric k-clustering. *IEEE Symposium on Foundations of Computer Science* (pp. 349–358).

Pelleg, D., & Moore, A. (1999). Accelerating exact k-means algorithms with geometric reasoning. *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 277–281). AAAI Press.

Pelleg, D., & Moore, A. (2000). X-means: Extending K-means with efficient estimation of the number of clusters. *Proceedings of the 17th International Conf. on Machine Learning* (pp. 727–734). Morgan Kaufmann, San Francisco, CA.

Peña, J., Lozano, J., & Larrañaga, P. (1999). An empirical comparison of four initialization methods for the k-means algorithm. *Pattern recognition letters*, *20*, 1027–1040.

Sand, P., & Moore, A. (2001). Repairing faulty mixture models using density estimation. *Proceedings of the 18th International Conf. on Machine Learning*. Morgan Kaufmann, San Francisco, CA.

Sand, P., & Moore, A. (2002). Fastmix clustering software. `http://www.cs.cmu.edu/~psand/`.

Zadeh, L. (1965). Fuzzy sets. *Information and Control*, *8*, 338–353.

Zhang, B. (2000). *Generalized k-harmonic means – boosting in unsupervised learning* (Technical Report HPL-2000-137). Hewlett-Packard Labs.

Zhang, B., Hsu, M., & Dayal, U. (1999). *K-harmonic means – a data clustering algorithm* (Technical Report HPL-1999-124). Hewlett-Packard Labs.

Zhang, T., Ramakrishnan, R., & Livny, M. (1997). BIRCH: a new data clustering algorithm and its applications (pp. 141–182.).

Zhao, Y., & Karypis, G. (2001). *Criterion functions for document clustering: experiments and analysis* (Technical Report 01-40).

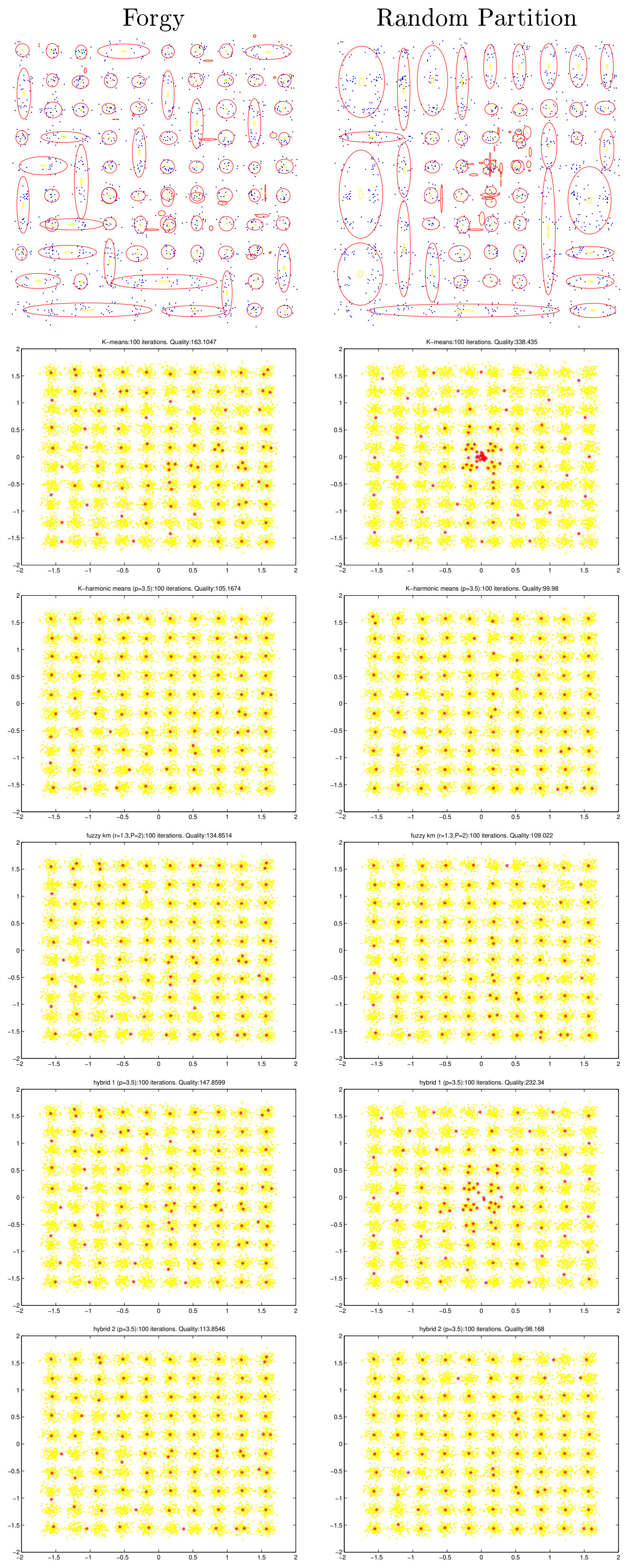

Figure 3. Experiment 1: Final results for the BIRCH dataset. From top to bottom: GEM, KM, KHM, FKM, H1, H2. The plot for GEM was generated by the FastMix software, showing the 1-sigma contours.