Chapter 2

K-Means

Joydeep Ghosh and Alexander Liu

Contents

2.1 Introduction

In this chapter, we describe the `k-means` algorithm, a straightforward and widely used clustering algorithm. Given a set of objects (records), the goal of clustering or segmentation is to divide these objects into groups or "clusters" such that objects within a group tend to be more similar to one another as compared to objects belonging to different groups. In other words, clustering algorithms place similar points in the same cluster while placing dissimilar points in different clusters. Note that, in contrast to *supervised* tasks such as regression or classification where there is a notion of a target value or class label, the objects that form the inputs to a clustering procedure do not come with an associated target. Therefore, *clustering* is often referred to as unsupervised learning. Because there is no need for labeled data, unsupervised algorithms are suitable for many applications where labeled data is difficult to obtain. Unsupervised tasks such as clustering are also often used to explore and characterize the dataset before running a supervised learning task. Since clustering makes no use of class labels, some notion of similarity must be defined based on the attributes of the objects. The definition of similarity and the method in which points are clustered differ based on the clustering algorithm being applied. Thus, different clustering algorithms are suited to different types of datasets and different purposes. The "best" clustering algorithm to use therefore depends on the application. It is not uncommon to try several different algorithms and choose depending on which is the most useful.

The k-means algorithm is a simple iterative clustering algorithm that partitions a given dataset into a user-specified number of clusters, k. The algorithm is simple to implement and run, relatively fast, easy to adapt, and common in practice. It is historically one of the most important algorithms in data mining.

Historically, k-means in its essential form has been discovered by several researchers across different disciplines, most notably by Lloyd (1957, 1982)[16],[1] Forgey (1965) [9], Friedman and Rubin (1967) [10], and McQueen (1967) [17]. A detailed history of k-means along with descriptions of several variations are given in Jain and Dubes [13]. Gray and Neuhoff [11] provide a nice historical background for k-means placed in the larger context of hill-climbing algorithms.

In the rest of this chapter, we will describe how k-means works, discuss the limitations of k-means, give some examples of k-means on artificial and real datasets, and briefly discuss some extensions to the k-means algorithm. We should note that our list of extensions to k-means is far from exhaustive, and the reader is encouraged to continue their own research on the aspect of k-means of most interest to them.

2.2 The k-means Algorithm

The k-means algorithm applies to objects that are represented by points in a d-dimensional vector space. Thus, it clusters a set of d-dimensional vectors, $D = \{\mathbf{x_i} | i = 1, \ldots, N\}$, where $\mathbf{x_i} \in \Re^d$ denotes the ith object or "data point." As discussed in the introduction, k-means is a clustering algorithm that partitions D into k clusters of points. That is, the k-means algorithm clusters all of the data points in D such that each point $\mathbf{x_i}$ falls in one and only one of the k partitions. One can keep track of which point is in which cluster by assigning each point a cluster ID. Points with the same cluster ID are in the same cluster, while points with different cluster IDs are in different clusters. One can denote this with a cluster membership vector $\mathbf{m}$ of length N, where m_i is the cluster ID of $\mathbf{x_i}$.

The value of k is an input to the base algorithm. Typically, the value for k is based on criteria such as prior knowledge of how many clusters actually appear in D, how many clusters are desired for the current application, or the types of clusters found by exploring/experimenting with different values of k. How k is chosen is not necessary for understanding how k-means partitions the dataset D, and we will discuss how to choose k when it is not prespecified in a later section.

In k-means, each of the k clusters is represented by a single point in $\Re^d$. Let us denote this set of cluster representatives as the set $C = \{\mathbf{c_j} | j = 1, \ldots, k\}$. These k cluster representatives are also called the *cluster means* or *cluster centroids*; we will discuss the reason for this after describing the k-means objective function.

[1] Lloyd first described the algorithm in a 1957 Bell Labs technical report, which was finally published in 1982.

In clustering algorithms, points are grouped by some notion of "closeness" or "similarity." In `k-means`, the default measure of closeness is the Euclidean distance. In particular, one can readily show that `k-means` attempts to minimize the following nonnegative cost function:

$$Cost = \sum_{i=1}^{N} (argmin_j ||\mathbf{x_i} - \mathbf{c_j}||_2^2) \tag{2.1}$$

In other words, `k-means` attempts to minimize the total squared Euclidean distance between each point $\mathbf{x_i}$ and its closest cluster representative $\mathbf{c_j}$. Equation 2.1 is often referred to as the `k-means` objective function.

The `k-means` algorithm, depicted in Algorithm 2.1, clusters D in an iterative fashion, alternating between two steps: (1) reassigning the cluster ID of all points in D and (2) updating the cluster representatives based on the data points in each cluster. The algorithm works as follows. First, the cluster representatives are initialized by picking k points in $\Re^d$. Techniques for selecting these initial seeds include sampling at random from the dataset, setting them as the solution of clustering a small subset of the data, or perturbing the global mean of the data k times. In Algorithm 2.1, we initialize by randomly picking k points. The algorithm then iterates between two steps until convergence.

Step 1: Data assignment. Each data point is assigned to its *closest* centroid, with ties broken arbitrarily. This results in a partitioning of the data.

Step 2: Relocation of "means." Each cluster representative is relocated to the center (i.e., arithmetic mean) of all data points assigned to it. The rationale of this step is based on the observation that, given a set of points, the single best representative for this set (in the sense of minimizing the sum of the squared Euclidean distances between each point and the representative) is nothing but the mean of the data points. This is also why the cluster representative is often interchangeably referred to as the *cluster mean* or *cluster centroid*, and where the algorithm gets its name from.

The algorithm converges when the assignments (and hence the $\mathbf{c_j}$ values) no longer change. One can show that the `k-means` objective function defined in Equation 2.1 will decrease whenever there is a change in the assignment or the relocation steps, and convergence is guaranteed in a finite number of iterations.

Note that each iteration needs $N \times k$ comparisons, which determines the time complexity of one iteration. The number of iterations required for convergence varies and may depend on N, but as a first cut, `k-means` can be considered linear in the dataset size. Moreover, since the comparison operation is linear in d, the algorithm is also linear in the dimensionality of the data.

Limitations. The greedy-descent nature of `k-means` on a nonconvex cost implies that the convergence is only to a local optimum, and indeed the algorithm is typically quite sensitive to the initial centroid locations. In other words,

Algorithm 2.1 The `k-means` algorithm

Input: Dataset D, number clusters k
Output: Set of cluster representatives C, cluster membership vector $\mathbf{m}$
/* Initialize cluster representatives C */
Randomly choose k data points from D
Use these k points as initial set of cluster representatives C
repeat
/* Data Assignment */
Reassign points in D to closest cluster mean
Update $\mathbf{m}$ such that m_i is cluster ID of ith point in D
/* Relocation of means */
Update C such that c_j is mean of points in jth cluster
until convergence of objective function $\sum_{i=1}^{N}(argmin_j||\mathbf{x_i} - \mathbf{c_j}||_2^2)$

initializing the set of cluster representatives C differently can lead to very different clusters, even on the same dataset D. A poor initialization can lead to very poor clusters. We will see an example of this in the next section when we look at examples of `k-means` applied to artificial and real data. The local minima problem can be countered to some extent by running the algorithm multiple times with different initial centroids and then selecting the best result, or by doing limited local search about the converged solution. Other approaches include methods such as those described in [14] that attempt to keep `k-means` from converging to local minima. [8] also contains a list of different methods of initialization, as well as a discussion of other limitations of `k-means`.

As mentioned, choosing the optimal value of k may be difficult. If one has knowledge about the dataset, such as the number of partitions that naturally comprise the dataset, then that knowledge can be used to choose k. Otherwise, one must use some other criteria to choose k, thus solving the *model selection* problem. One naive solution is to try several different values of k and choose the clustering which minimizes the `k-means` objective function (Equation 2.1). Unfortunately, the value of the objective function is not as informative as one would hope in this case. For example, the cost of the optimal solution decreases with increasing k till it hits zero when the number of clusters equals the number of distinct data points. This makes it more difficult to use the objective function to (a) directly compare solutions with different numbers of clusters and (b) find the optimum value of k. Thus, if the desired k is not known in advance, one will typically run `k-means` with different values of k, and then use some other, more suitable criterion to select one of the results. For example, SAS uses the cube-clustering criterion, while X-means adds a complexity term (which increases with k) to the original cost function (Equation 2.1) and then identifies the k which minimizes this adjusted cost [20]. Alternatively, one can progressively increase the number of clusters, in conjunction with a suitable stopping criterion. Bisecting `k-means` [21] achieves this by first putting all the data into a single cluster, and then recursively splitting the least compact cluster into two clusters using 2-means. The

celebrated LBG algorithm [11] used for vector quantization doubles the number of clusters till a suitable code-book size is obtained. Both these approaches thus alleviate the need to know k beforehand. Many other researchers have studied this problem, such as [18] and [12].

In addition to the above limitations, `k-means` suffers from several other problems that can be understood by first noting that the problem of fitting data using a mixture of k Gaussians with identical, isotropic covariance matrices ($\Sigma = \sigma^2\mathbf{I}$), where $\mathbf{I}$ is the identity matrix, results in a "soft" version of `k-means`. More precisely, if the soft assignments of data points to the mixture components of such a model are instead hardened so that each data point is solely allocated to the most likely component [3], then one obtains the `k-means` algorithm. From this connection it is evident that `k-means` inherently assumes that the dataset is composed of a mixture of k balls or hyperspheres of data, and each of the k clusters corresponds to one of the mixture components. Because of this implicit assumption, `k-means` will falter whenever the data is not well described by a superposition of reasonably separated spherical Gaussian distributions. For example, `k-means` will have trouble if there are non-convex-shaped clusters in the data. This problem may be alleviated by rescaling the data to "whiten" it before clustering, or by using a different distance measure that is more appropriate for the dataset. For example, information-theoretic clustering uses the KL-divergence to measure the distance between two data points representing two discrete probability distributions. It has been recently shown that if one measures distance by selecting any member of a very large class of divergences called *Bregman divergences* during the assignment step and makes no other changes, the essential properties of `k-means`, including guaranteed convergence, linear separation boundaries, and scalability, are retained [1]. This result makes `k-means` effective for a much larger class of datasets so long as an appropriate divergence is used.

Another method of dealing with nonconvex clusters is by pairing `k-means` with another algorithm. For example, one can first cluster the data into a large number of groups using `k-means`. These groups are then agglomerated into larger clusters using single link hierarchical clustering, which can detect complex shapes. This approach also makes the solution less sensitive to initialization, and since the hierarchical method provides results at multiple resolutions, one does not need to worry about choosing an exact value for k either; instead, one can simply use a large value for k when creating the initial clusters.

The algorithm is also sensitive to the presence of outliers, since "mean" is not a robust statistic. A preprocessing step to remove outliers can be helpful. Postprocessing the results, for example, to eliminate small clusters, or to merge close clusters into a large cluster, is also desirable. Ball and Hall's ISODATA algorithm from 1967 effectively used both pre- and postprocessing on `k-means`.

Another potential issue is the problem of "empty" clusters [4]. When running `k-means`, particularly with large values of k and/or when data resides in very high dimensional space, it is possible that at some point of execution, there exists a cluster representative c_j such that all points x_i in D are closer to some other cluster representative that is not c_j. When points in D are assigned to their closest cluster, the jth cluster will have zero points assigned to it. That is, cluster j is now an empty cluster.

The standard algorithm does not guard against empty clusters, but simple extensions (such as reinitializing the cluster representative of the empty cluster or "stealing" some points from the largest cluster) are possible.

2.3 Available Software

Because of the `k-means` algorithm's simplicity, effectiveness, and historical importance, software to run the k-means algorithm is readily available in several forms. It is a standard feature in many popular data mining software packages. For example, it can be found in Weka or in SAS under the FASTCLUS procedure. It is also commonly included as add-ons to existing software. For example, several implementations of `k-means` are available as parts of various toolboxes in MATLAB®. `k-means` is also available in Microsoft Excel after adding XLMiner. Finally, several stand-alone versions of `k-means` exist and can be easily found on the Internet.

The algorithm is also straightforward to code, and the reader is encouraged to create their own implementation of `k-means` as an exercise.

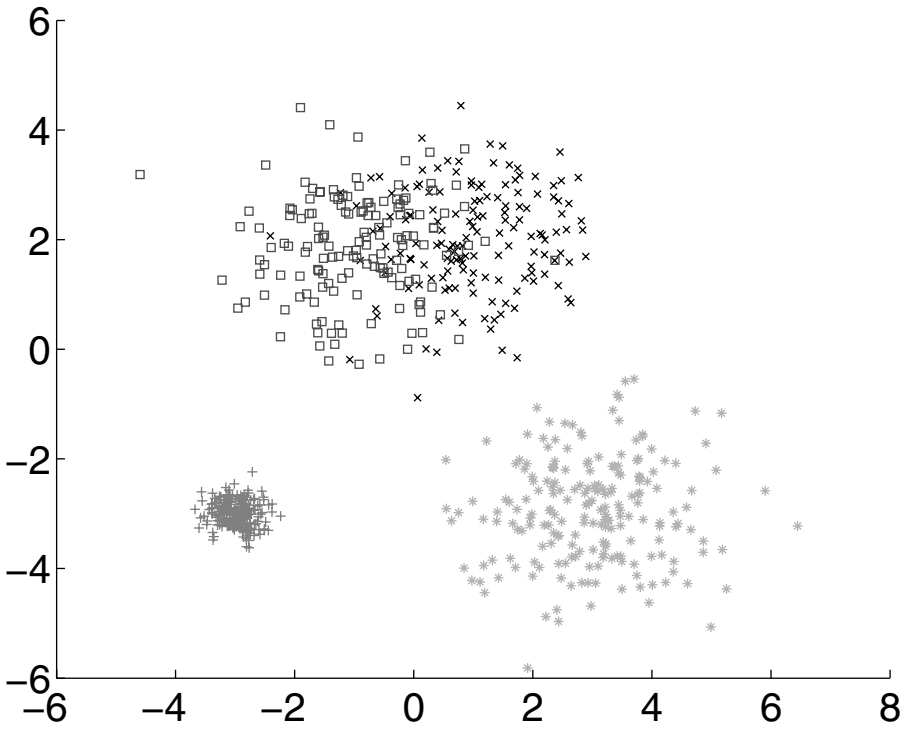

Figure 2.1 The artificial dataset used in our example; the data is drawn from a mixture of four Gaussians.

2.4 Examples

Let us first show an example of `k-means` on an artificial dataset to illustrate how `k-means` works. We will use artificial data drawn from four 2-D Gaussians and use a value of $k = 4$; the dataset is illustrated in Figure 2.1. Data drawn from a particular Gaussian is plotted in the same color in Figure 2.1. The blue data consists of 200 points drawn from a Gaussian with mean at $(-3, -3)$ and covariance matrix $.0625 \times \mathbf{I}$, where $\mathbf{I}$ is the identity matrix. The green data consists of 200 points drawn from a Gaussian with mean at $(3, -3)$ and covariance matrix $\mathbf{I}$. Finally, we have overlapping yellow and red data drawn from two nearby Gaussians. The yellow data consists of 150 points drawn from a Gaussian with mean $(-1, 2)$ and covariance matrix $\mathbf{I}$, while the red data consists of 150 points drawn from a Gaussian with mean (1,2) and covariance matrix $\mathbf{I}$. Despite the overlap between the red and yellow points, one would expect `k-means` to do well since we do have the right value of k and the data is generated by a mixture of spherical Gaussians, thus matching nicely with the underlying assumptions of the algorithm.

The first step in `k-means` is to initialize the cluster representatives. This is illustrated in Figure 2.2a, where k points in the dataset have been picked randomly. In this figure and the following figures, the cluster means C will be represented by a large colored circle with a black outline. The color corresponds to the cluster ID of that particular cluster, and all points assigned to that cluster are represented as points of the same color. These colors have no definite connection with the colors in Figure 2.1 (see Exercise 7). Since points have not been assigned cluster IDs in Figure 2.2a, they are plotted in black.

The next step is to assign all points to their closest cluster representative; this is illustrated in Figure 2.2b, where each point has been plotted to match the color of its closest cluster representative. The third step in `k-means` is to update the k cluster representatives to correspond to the mean of all points currently assigned to that cluster. This step is illustrated in Figure 2.2c. In particular, we have plotted the old cluster representatives with a black "X" symbol and the new, updated cluster representatives as a large colored circle with a black outline. There is also a line connecting the old cluster mean with the new, updated cluster mean. One can observe that the cluster representatives have moved to reflect the current centroids of each cluster.

The `k-means` algorithm now iterates between two steps until convergence: reassigning points in D to their closest cluster representative and updating the k cluster representatives. We have illustrated the first four iterations of `k-means` in Figures 2.2 and 2.3. The final clusters after convergence are shown in Figure 2.3d. Note that this example took eight iterations to converge. Visually, however, there is little change in the diagrams between iterations 4 and 8, and these pictures are omitted for space reasons. As one can see by comparing Figure 2.3d with Figure 2.1, the clusters found by `k-means` match well with the true, underlying distribution.

In the previous section, we mentioned that `k-means` is sensitive to the initial points picked as clusters. In Figure 2.4, we show what happens when the k representatives are

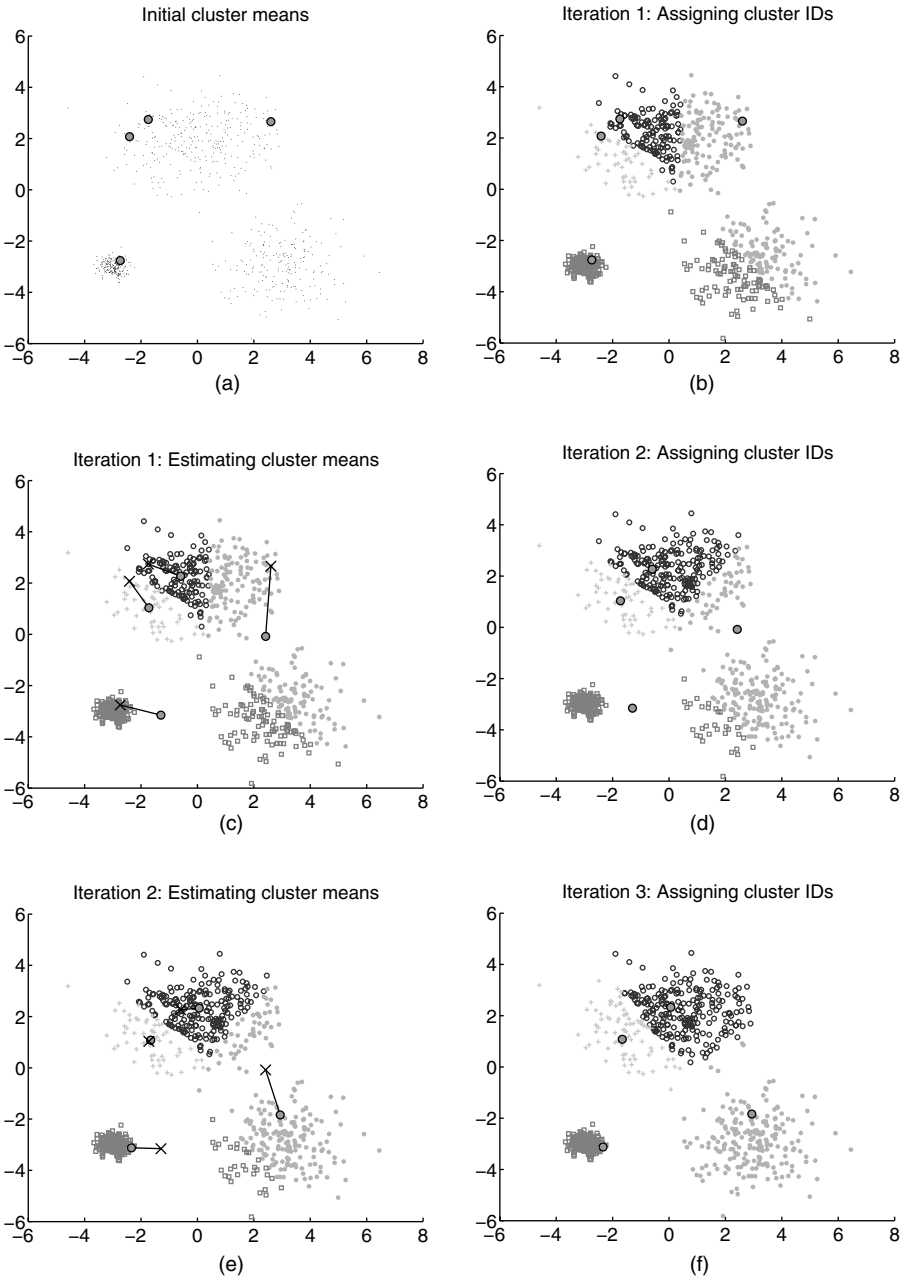

Figure 2.2 `k-means` on artificial data.

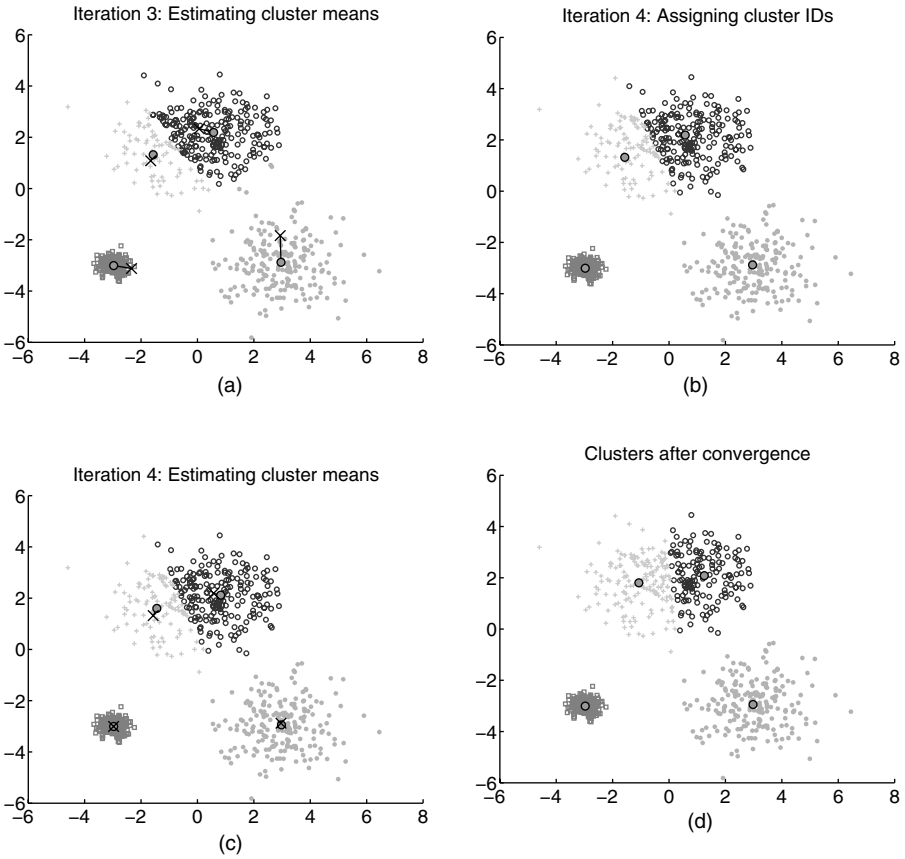

Figure 2.3 `k-means` on artificial data. (Continued from Figure 2.2.)

initialized poorly on the same artificial dataset used in Figures 2.2 and 2.3. Figures 2.4a and c show two initializations that lead to poor clusters in Figures 2.4b and d. These results are considered poor since they do not correspond well to the true underlying distribution.

Finally, let us examine the performance of `k-means` on a simple, classic benchmark dataset. In our example, we use the Iris dataset (available from the UCI data mining repository), which contains 150 data points from three classes. Each class represents a different species of the Iris flower, and there are 50 points from each class. While there are four dimensions (representing sepal width, sepal length, petal width, and petal length), only two dimensions (petal width and petal length) are necessary to discriminate the three classes. The Iris dataset is plotted in Figure 2.5a along the dimensions of petal width and petal length.

In Figure 2.5b, we show an example of the `k-means` algorithm run on the Iris dataset with $k = 3$, using only the attributes of petal width and petal length. The

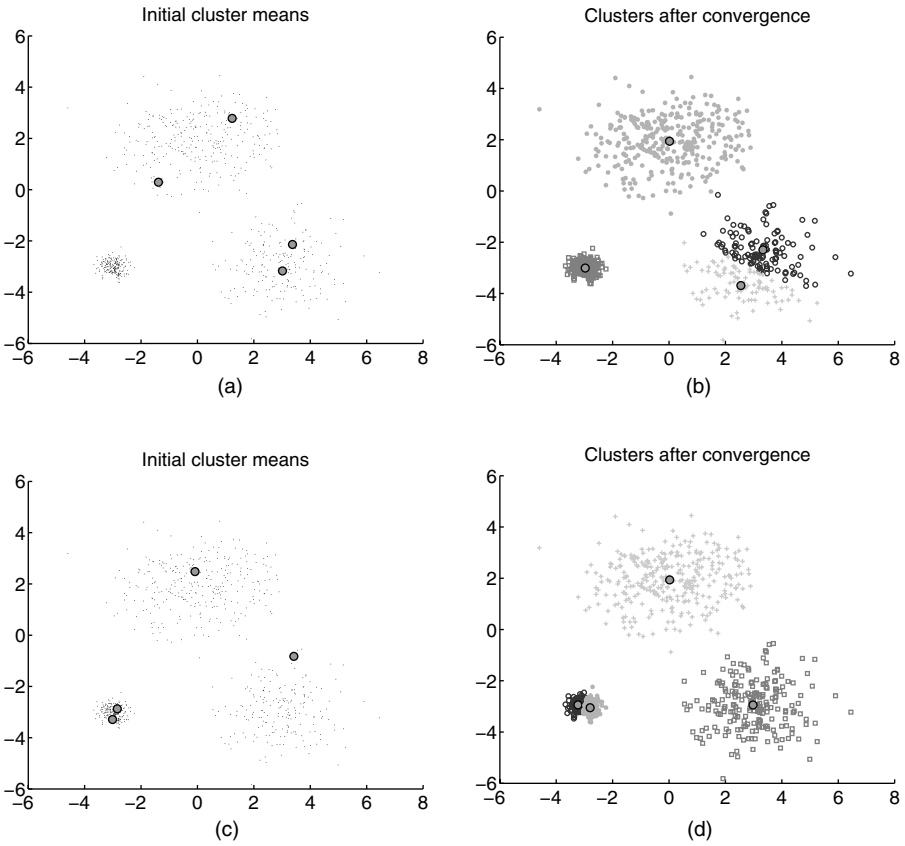

Figure 2.4 Examples of poor clustering after poor initialization; these resultant clusters are considered "poor" in the sense that they do not match well with the true, underlying distribution.

`k-means` algorithm is able to cluster the data points such that each cluster is composed mostly of flowers from the same species.

2.5 Advanced Topics

In this section, we discuss some generalizations, connections, and extensions that have been made to the `k-means` algorithm. However, we should note that this section is far from exhaustive. Research on `k-means` has been extensive and is still active. Instead, the goal of this section is to complement some of the previously discussed issues regarding `k-means`.

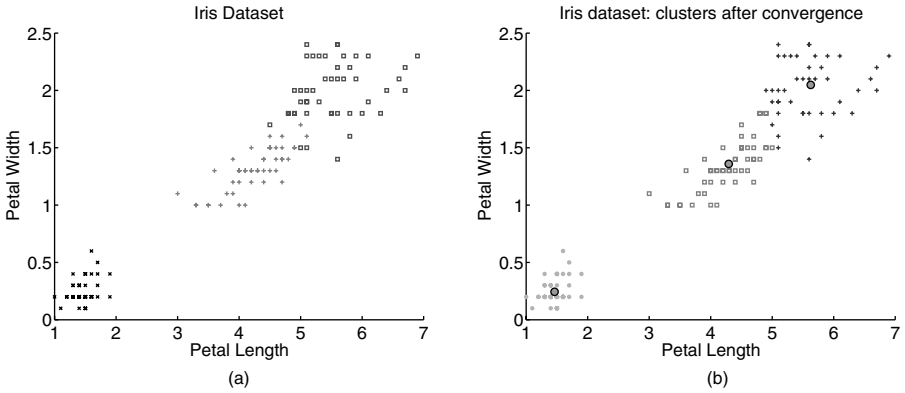

Figure 2.5 (a) Iris dataset; each color is a different species of Iris; (b) Result of `k-means` on Iris dataset; each color is a different cluster; note that there is not necessarily a correspondence between colors in (a) and (b) (see Exercise 7).

As mentioned earlier, `k-means` is closely related to fitting a mixture of k isotropic Gaussians to the data. Moreover, the generalization of the distance measure to all Bregman divergences is related to fitting the data with a mixture of k components from the exponential family of distributions. Another broad generalization is to view the "means" as probabilistic models instead of points in R^d. Here, in the assignment step, each data point is assigned to the model most likely to have generated it. In the "relocation" step, the model parameters are updated to best fit the assigned datasets. Such *model-based* `k-means` [23] allow one to cater to more complex data, for example, sequences described by Hidden Markov models.

One can also "kernelize" `k-means` [5]. Though boundaries between clusters are still linear in the implicit high-dimensional space, they can become nonlinear when projected back to the original space, thus allowing kernel `k-means` to deal with more complex clusters. Dhillon et al. [5] have shown a close connection between kernel `k-means` and spectral clustering. The *K-medoid* [15] algorithm is similar to `k-means`, except that the centroids have to belong to the dataset being clustered. Fuzzy c-means [6] is also similar, except that it computes fuzzy membership functions for each cluster rather than a hard one.

To deal with very large datasets, substantial effort has also gone into further speeding up `k-means`, most notably by using kd-trees [19] or exploiting the triangular inequality [7] to avoid comparing each data point with all the centroids during the assignment step.

Finally, we discuss two straightforward extensions of `k-means`. The first is a variant of `k-means` called `soft k-means`. In the standard `k-means` algorithm, each point x_i belongs to one and only one cluster. In `soft k-means`, this constraint is relaxed, and each point x_i can belong to each cluster with some unknown probability. In `soft k-means`, for each point x_i, one maintains a set of k probabilities or weights

that describe the likelihood that x_i belongs to each cluster. These weights are based on the distance of x_i to each of the cluster representatives C, where the probability that x_i is from cluster j is proportional to the similarity between x_i and c_j. The cluster representatives in this case are found by taking the expected value of the cluster mean over all points in the dataset D.

The second extension of `k-means` deals with semisupervised learning. In the introduction, we made a distinction between supervised learning and unsupervised learning. In brief, supervised learning makes use of class labels while unsupervised learning does not. The `k-means` algorithm is a purely unsupervised algorithm. There also exists a category of learning algorithms called *semisupervised algorithms*. Semisupervised learning algorithms are capable of making use of both labeled and unlabeled data. Semisupervised learning is a useful compromise between purely supervised methods and purely unsupervised methods. Supervised learning methods typically require very large amounts of labeled data; semisupervised methods are useful when very few labeled examples are available. Unsupervised learning methods, which do not look at class labels, may learn models inappropriate for the application at hand. When running `k-means`, one has no control over the final clusters that are discovered; these clusters may or may not correspond well to some underlying concept that one is interested in. For example, in Figure 2.5b, a poor initialization may have resulted in clusters which do not correspond well to the Iris species in the dataset. Semisupervised methods, which can take guidance in the form of labeled points, are more likely to create clusters which correspond to a given set of class labels.

Research into semisupervised variants of `k-means` include [22] and [2]. One of the algorithms from [2] called `seeded k-means` is a simple extension to `k-means` that uses labeled data to help initialize the value of k and the cluster representatives C. In this approach, k is chosen to be the same as the number of classes in the labeled data, while c_j is initialized as the mean of all labeled points in the jth class. Note that, unlike unsupervised `k-means`, there is now a known correspondence between the jth cluster and the jth class. After initialization, `seeded k-means` iterates over the same two steps as `k-means` (updating cluster memberships and updating cluster means) until convergence.

2.6 Summary

The `k-means` algorithm is a simple iterative clustering algorithm that partitions a dataset into k clusters. At its core, the algorithm works by iterating over two steps: (1) clustering all points in the dataset based on the distance between each point and its closest cluster representative and (2) reestimating the cluster representatives. Limitations of the `k-means` algorithm include the sensitivity of `k-means` to initialization and determining the value of k.

Despite its drawbacks, `k-means` remains the most widely used partitional clustering algorithm in practice. The algorithm is simple, easily understandable, and

reasonably scalable, and can be easily modified to deal with different scenarios such as semisupervised learning or streaming data. Continual improvements and generalizations of the basic algorithm have ensured its continued relevance and gradually increased its effectiveness as well.

2.7 Exercises

1. Using the standard benchmark Iris dataset (available online from the UCI dataset repository), run `k-means` to obtain results similar to Figure 2.5b. It is sufficient to look at only the attributes of "petal width" and "petal length."

   What happens when one uses a value for k other than three? How do different cluster initializations affect the final clusters? Why are these results potentially different than the results given in Figure 2.5b?
2. Prove that the value of the `k-means` objective function converges when `k-means` is run.
3. Describe three advantages and three disadvantages of `k-means` compared to other clustering methods (e.g., agglomerative clustering).
4. Describe or plot a two-dimensional example where `k-means` would be unsuitable for finding clusters.
5. In `k-means`, after the cluster means have converged, what is the shape of the cluster boundaries? How is this related to Voronoi tesselations?
6. Does `k-means` guarantee that points within the same cluster are more similar than points from different clusters? That is, prove or disprove that, after `k-means` has converged, the squared Euclidean distance between two points in the same cluster is always less than the squared Euclidean distance between two points from different clusters.
7. Assume one is given a hypothetical dataset D consisting of 10 points. `k-means` is run twice on this dataset. Let us denote the cluster IDs of the 10 points in D as a vector $\mathbf{m}$, where m_i, the ith entry in the vector, is the cluster ID of the ith point in D.

   The cluster IDs of the 10 points from the first time `k-means` is run are $\mathbf{m^1} = [1, 1, 1, 2, 2, 2, 3, 3, 3, 3]$, while the cluster IDs obtained from the second run of `k-means` are $\mathbf{m^2} = [3, 3, 3, 1, 1, 1, 2, 2, 2, 2]$.

   What is the difference between the two sets of cluster IDs? Do the actual cluster IDs of the points in D mean anything? What does this imply when comparing the results of different clustering algorithms? What does this imply when comparing the results of clustering algorithms with known class labels?
8. Create your own implementation of `k-means` and a method of creating artificial data drawn from k Gaussian distributions. Test your code on the artificial data and keep track of how many iterations it takes for `k-means` to converge.

9. Using the code generated in the previous exercise, plot the average distance of each point from its cluster mean versus the number of clusters k. Is the average distance of a point from its cluster mean a good method of automatically determining the number of clusters k? Why or why not? What can potentially happen when the number of clusters k is equal to the number of points in the dataset?
10. Research and describe an extension to the standard `k-means` algorithm. Depending on individual interests, this could include recent work on making `k-means` more computationally efficient, work on extending `k-means` to semisupervised learning, work on adapting other distance metrics into `k-means`, or many other possibilities.

References

[1] A. Banerjee, S. Merugu, I. Dhillon, and J. Ghosh. "Clustering with Bregman divergences," *Journal of Machine Learning Research* (*JMLR*), vol. 6, pp. 1705–1749, 2005.

[2] S. Basu, A. Banerjee, and R. Mooney. "Semi-supervised clustering by seeding," *International Conference on Machine Learning 2002*, pp. 27–34, 2002.

[3] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 2006.

[4] P. S. Bradley, K. P. Bennett, and A. Demiriz. "Constrained k-means clustering," *Technical Report MSR-TR-2000-65*, 2000.

[5] I. S. Dhillon, Y. Guan, and B. Kulis. "Kernel k-means: Spectral clustering and normalized cuts," *KDD 2004*, pp. 551–556, 2004.

[6] J. C. Dunn. "A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters," *Journal of Cybernetics*, vol. 3, pp. 32–57, 1974.

[7] C. Elkan. "Using the triangle inequality to accelerate k-means," *International Conference on Machine Learning 2003*, pp. 147–153, 2003.

[8] C. Elkan. "Clustering with k-means: Faster, smarter, cheaper," *Keynote talk at Workshop on Clustering High-Dimensional Data, SIAM International Conference on Data Mining*, 2004.

[9] E. Forgey. "Cluster analysis of multivariate data: Efficiency vs. interpretability of classification," *Biometrics*, 21, pp. 768, 1965.

[10] H. P. Friedman and J. Rubin. "On some invariant criteria for grouping data," *Journal of American Statistical Association*, 62, pp. 1159–1178, 1967.

[11] R. M. Gray and D. L. Neuhoff. "Quantization," *IEEE Transactions on Information Theory,* vol. 44, no. 6, pp. 2325–2384, 1998.

[12] G. Hamerly and C. Elkan. "Learning the k in k-means," *Neural Information Processing Systems*, 2003.

[13] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*, Prentice Hall, 1988.

[14] T. Kanungo, D. M. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu. "A local search approximation algorithm for k-means clustering," *Computational Geometry: Theory and Applications*, 28 (2004), pp. 89–112, 2004.

[15] L. Kaufman and P. J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*, 1990.

[16] S. P. Lloyd. "Least squares quantization in PCM," unpublished Bell Lab. Tech. Note, portions presented at the Institute of Mathematical Statistics Meet., Atlantic City, NJ, Sept. 1957. Also, *IEEE Trans. Inform. Theory* (Special Issue on Quantization), vol. IT-28, pp. 129–137, Mar. 1982.

[17] J. McQueen. "Some methods for classification and analysis of mutivariate observations," *Proc. 5th Berkeley Symp. Math., Statistics and Probability*, 1, pp. 281–296, 1967.

[18] G. W. Milligan. "Clustering validation: Results and implications for applied analyses," *Clustering and Classification*, P. Arabie, L. J. Hubery, and G. De Soete, ed., pp. 341–375, 1996.

[19] D. Pelleg and A. Moore. "Accelerating exact k-means algorithms with geometric reasoning," *KDD 1999*, pp. 227–281, 1999.

[20] D. Pelleg and A. Moore. "X-means: Extending k-means with efficient estimation of the number of clusters," *International Conference on Machine Learning 2000*, pp. 727–734, 2000.

[21] M. Steinbach, G. Karypis, and V. Kumar. "A comparison of document clustering techniques," *Proc. KDD Workshop on Text Mining*, 2000.

[22] K. Wagstaff, C. Cardie, S. Rogers, S. Schrödl. "Constrained k-means clustering with background knowledge," *International Conference on Machine Learning 2001*, pp. 577–584, 2001.

[23] S. Zhong and J. Ghosh. "A unified framework for model-based clustering," *Journal of Machine Learning Research* (*JMLR*), vol. 4, pp. 1001–1037, 2003.