Optimizing the group representatives with the assignment fixed. Now we turn to the problem of choosing the group representatives, with the clustering (group assignments) fixed, in order to minimize our objective J^{clust}. It turns out that this problem also has a simple and natural solution.

We start by re-arranging the sum of N terms into k sums, each associated with one group. We write

$$J^{\text{clust}} = J_1 + \cdots + J_k,$$

where

$$J_j = (1/N) \sum_{i \in G_j} \|x_i - z_j\|^2$$

is the contribution to the objective J^{clust} from the vectors in group j. (The sum here means that we should add up all terms of the form $\|x_i - z_j\|^2$, for any $i \in G_j$, *i.e.*, for any vector x_i in group j; see appendix A.)

The choice of group representative z_j only affects the term J_j; it has no effect on the other terms in J^{clust}. So we can choose each z_j to minimize J_j. Thus we should choose the vector z_j so as to minimize the mean square distance to the vectors in group j. This problem has a very simple solution: We should choose z_j to be the average (or mean or centroid) of the vectors x_i in its group:

$$z_j = (1/|G_j|) \sum_{i \in G_j} x_i,$$

where $|G_j|$ is standard mathematical notation for the number of elements in the set G_j, *i.e.*, the size of group j. (See exercise 4.1.)

So if we fix the group assignments, we minimize J^{clust} by choosing each group representative to be the average or centroid of the vectors assigned to its group. (This is sometimes called the *group centroid* or *cluster centroid*.)

4.3 The k-means algorithm

It might seem that we can now solve the problem of choosing the group assignments and the group representatives to minimize J^{clust}, since we know how to do this when one or the other choice is fixed. But the two choices are circular, *i.e.*, each depends on the other. Instead we rely on a very old idea in computation: We simply *iterate* between the two choices. This means that we repeatedly alternate between updating the group assignments, and then updating the representatives, using the methods developed above. In each step the objective J^{clust} gets better (*i.e.*, goes down) unless the step does not change the choice. Iterating between choosing the group representatives and choosing the group assignments is the celebrated *k-means algorithm* for clustering a collection of vectors.

The k-means algorithm was first proposed in 1957 by Stuart Lloyd, and independently by Hugo Steinhaus. It is sometimes called the Lloyd algorithm. The name 'k-means' has been used since the 1960s.

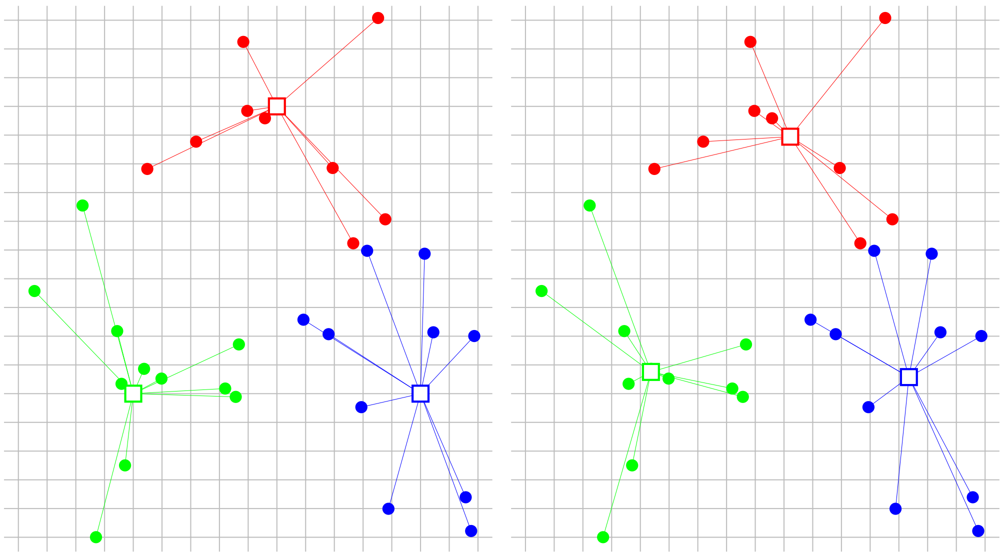

Figure 4.2 *One iteration of the k-means algorithm.* The 30 2-vectors x_i are shown as filled circles, and the 3 group representatives z_j are shown as rectangles. In the left-hand figure the vectors are each assigned to the closest representative. In the right-hand figure, the representatives are replaced by the cluster centroids.

Algorithm 4.1 k-MEANS ALGORITHM

given a list of N vectors $x_1, \ldots, x_N$, and an initial list of k group representative vectors $z_1, \ldots, z_k$

repeat until convergence

1. *Partition the vectors into k groups.* For each vector $i = 1, \ldots, N$, assign x_i to the group associated with the nearest representative.
2. *Update representatives.* For each group $j = 1, \ldots, k$, set z_j to be the mean of the vectors in group j.

One iteration of the k-means algorithm is illustrated in figure 4.2.

Comments and clarifications.

- Ties in step 1 can be broken by assigning x_i to the group associated with one of the closest representatives with the smallest value of j.
- It is possible that in step 1, one or more of the groups can be empty, *i.e.*, contain no vectors. In this case we simply drop this group (and its representative). When this occurs, we end up with a partition of the vectors into fewer than k groups.

- If the group assignments found in step 1 are the same in two successive iterations, the representatives in step 2 will also be the same. It follows that the group assignments and group representatives will never change in future iterations, so we should stop the algorithm. This is what we mean by 'until convergence'. In practice, one often stops the algorithm earlier, as soon as the improvement in J^{clust} in successive iterations becomes very small.

- We start the algorithm with a choice of initial group representatives. One simple method is to pick the representatives randomly from the original vectors; another is to start from a random assignment of the original vectors to k groups, and use the means of the groups as the initial representatives. (There are more sophisticated methods for choosing an initial representatives, but this topic is beyond the scope of this book.)

Convergence. The fact that J^{clust} decreases in each step implies that the k-means algorithm converges in a finite number of steps. However, depending on the initial choice of representatives, the algorithm can, and does, converge to different final partitions, with different objective values.

The k-means algorithm is a *heuristic*, which means it cannot guarantee that the partition it finds minimizes our objective J^{clust}. For this reason it is common to run the k-means algorithm several times, with different initial representatives, and choose the one among them with the smallest final value of J^{clust}. Despite the fact that the k-means algorithm is a heuristic, it is very useful in practical applications, and very widely used.

Figure 4.3 shows a few iterations generated by the k-means algorithm, applied to the example of figure 4.1. We take $k = 3$ and start with randomly chosen group representatives. The final clustering is shown in figure 4.4. Figure 4.5 shows how the clustering objective decreases in each step.

Interpreting the representatives. The representatives $z_1, \ldots, z_k$ associated with a clustering are quite interpretable. Suppose, for example, that voters in some election can be well clustered into 7 groups, on the basis of a data set that includes demographic data and questionnaire or poll data. If the 4th component of our vectors is the age of the voter, then $(z_3)_4 = 37.8$ tells us that the average age of voters in group 3 is 37.8. Insight gained from this data can be used to tune campaign messages, or choose media outlets for campaign advertising.

Another way to interpret the group representatives is to find one or a few of the original data points that are closest to each representive. These can be thought of as archetypes for the group.

Choosing k. It is common to run the k-means algorithm for different values of k, and compare the results. How to choose a value of k among these depends on how the clustering will be used, which we discuss a bit more in §4.5. But some general statements can be made. For example, if the value of J^{clust} with $k = 7$ is quite a bit smaller than the values of J^{clust} for $k = 2, \ldots, 6$, and not much larger than the values of J^{clust} for $k = 8, 9, \ldots$, we could reasonably choose $k = 7$, and conclude that our data (list of vectors) partitions nicely into 7 groups.

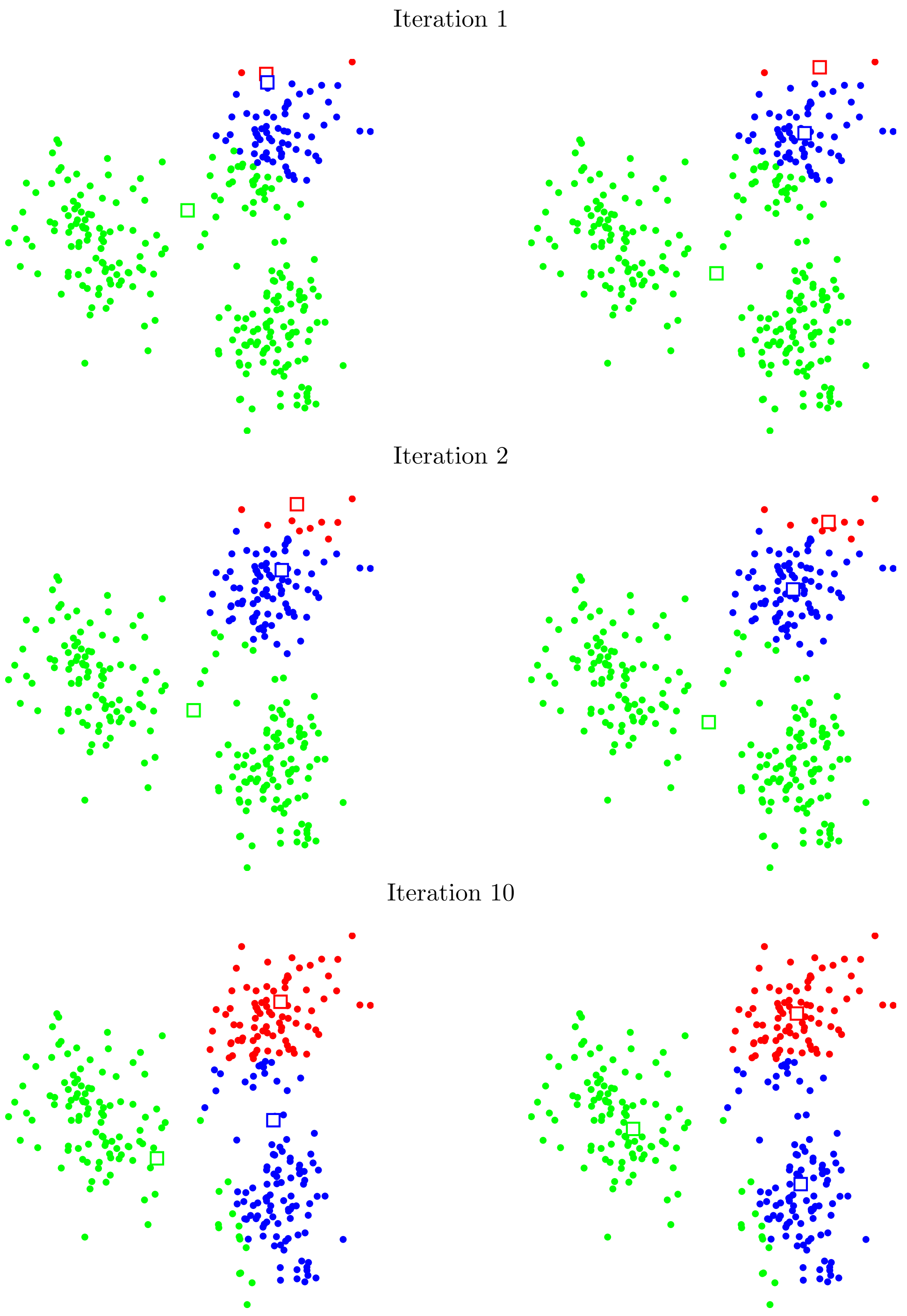

Figure 4.3 Three iterations of the k-means algorithm. The group representatives are shown as squares. In each row, the left-hand plot shows the result of partitioning the vectors in the 3 groups (step 1 of algorithm 4.1). The right-hand plot shows the updated representatives (step 2 of the algorithm).

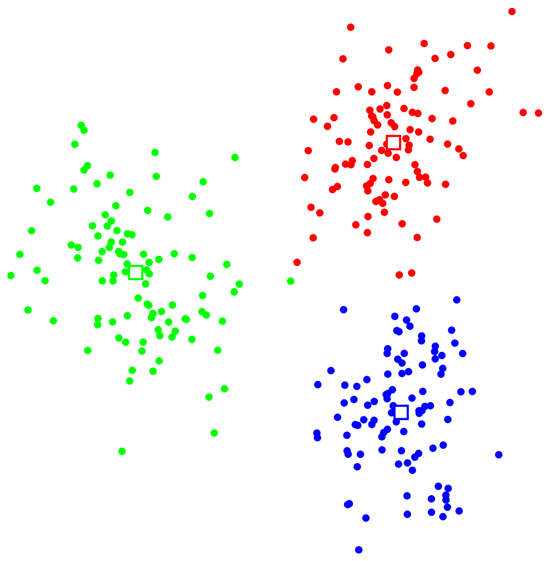

Figure 4.4 Final clustering.

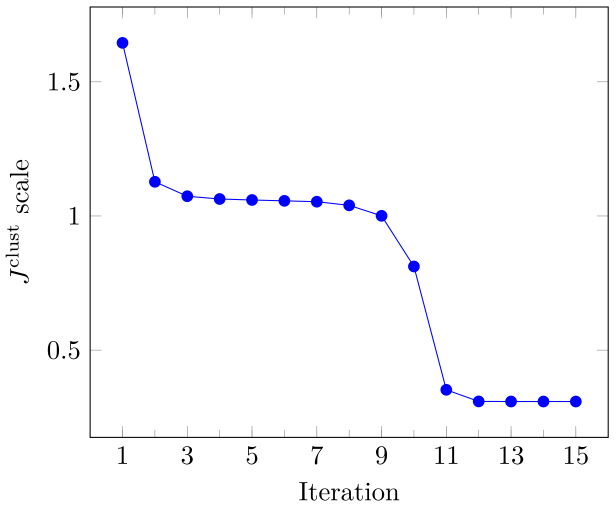

Figure 4.5 The clustering objective J^{clust} after step 1 of each iteration.

Complexity. In step 1 of the k-means algorithm, we find the nearest neighbor to each of N n-vectors, over the list of k centroids. This requires approximately $3Nkn$ flops. In step 2 we average the n-vectors over each of the cluster groups. For a cluster with p vectors, this requires $n(p-1)$ flops, which we approximate as np flops; averaging all clusters requires a total of Nn flops. This is less than the cost of partitioning in step 1. So k-means requires around $(3k+1)Nn$ flops per iteration. Its order is Nkn flops.

Each run of k-means typically takes fewer than a few tens of iterations, and usually k-means is run some modest number of times, like 10. So a very rough guess of the number of flops required to run k-means 10 times (in order to choose the best partition found) is $1000Nkn$ flops.

As an example, suppose we use k-means to partition $N = 100000$ vectors with size $n = 100$ into $k = 10$ groups. On a 1 Gflop/s computer we guess that this will take around 100 seconds. Given the approximations made here (for example, the number of iterations that each run of k-means will take), this is obviously a crude estimate.

4.4 Examples

4.4.1 Image clustering

The MNIST (Mixed National Institute of Standards) database of handwritten digits is a data set containing $N = 60000$ grayscale images of size 28×28, which we represent as n-vectors with $n = 28 \times 28 = 784$. Figure 4.6 shows a few examples from the data set. (The data set is available from Yann LeCun at `yann.lecun.com/exdb/mnist`.)

We use the k-means algorithm to partition these images into $k = 20$ clusters, starting with a random assignment of the vectors to groups, and repeating the experiment 20 times. Figure 4.7 shows the clustering objective versus iteration number for three of the 20 initial assignments, including the two that gave the lowest and the highest final values of the objective.

Figure 4.8 shows the representatives with the lowest final value of the clustering objective. Figure 4.9 shows the set with the highest value. We can see that most of the representatives are recognizable digits, with some reasonable confusion, for example between '4' and '9' or '3' and '8'. This is impressive when you consider that the k-means algorithm knows nothing about digits, handwriting, or even that the 784-vectors represent 28×28 images; it uses only the distances between 784-vectors. One interpretation is that the k-means algorithm has 'discovered' the digits in the data set.