Part II

Unsupervised Learning

CHAPTER

10 Clustering

Previous chapters have dealt with *supervised* learning problems, in which the basic goal is to learn a mapping from observable features x to a target y. In *unsupervised learning*, the goal is to directly understand the structure of the data x, without any specific target y to focus our interest.

Most unsupervised learning methods can be thought of as a form of data compression: if we are able to understand the data well enough, we should be able to summarize it accurately – organize the data in a meaningful way (so that similar data are grouped together), and/or provide a "shorter" description that would allow us to reconstruct a good approximation of the data.

Clustering methods for unsupervised learning attempt to describe the underlying structure of the data in terms of discrete groups, or *clusters*. Then, we can characterize the groups to provide a summary of the data. This characterization might be done by examining the data themselves (since if all the data in a given cluster are similar, we can just look at a few example points to get a sense of that group of data), or by reporting meaningful summary statistics of the cluster (for example, its mean value and spread).

Measuring Similarity

A fundamental difficulty in clustering is that the concept of a meaningful "group" of data can mean different things in different settings, and is not always clear-cut. When we as people look at a scatterplot of data points, we often perceive patterns and groups that allow us to understand or describe the data. But, the exact nature of these patterns can vary. Figure 10.1 shows several examples of data that form visual patterns in different ways. The most obvious of these, in Figure 10.1(a), has sets of data points that are all close to one another, but far from the other groups, so that the clusters differ significantly in their location in feature space. Figure 10.1(b) shows a more complex scenario, in which the two groups are not so far apart, but clearly form two shapes that are visible as contiguous regions with dense data. Figure 10.1(c) shows yet another example, in which some data are very densely sampled in one part of the space, and the rest are more sparsely scattered.

The key point here is that, while these groups are clear visually, it is less obvious what quantitative definition of a group would correctly identify these, and other, patterns. In general, our data will be high dimensional (many observed features), and we will not be able to check the clustering using a visual examination. Thus, whatever definition of similarity we adopt will, in some sense, determine the groupings that we obtain from clustering.

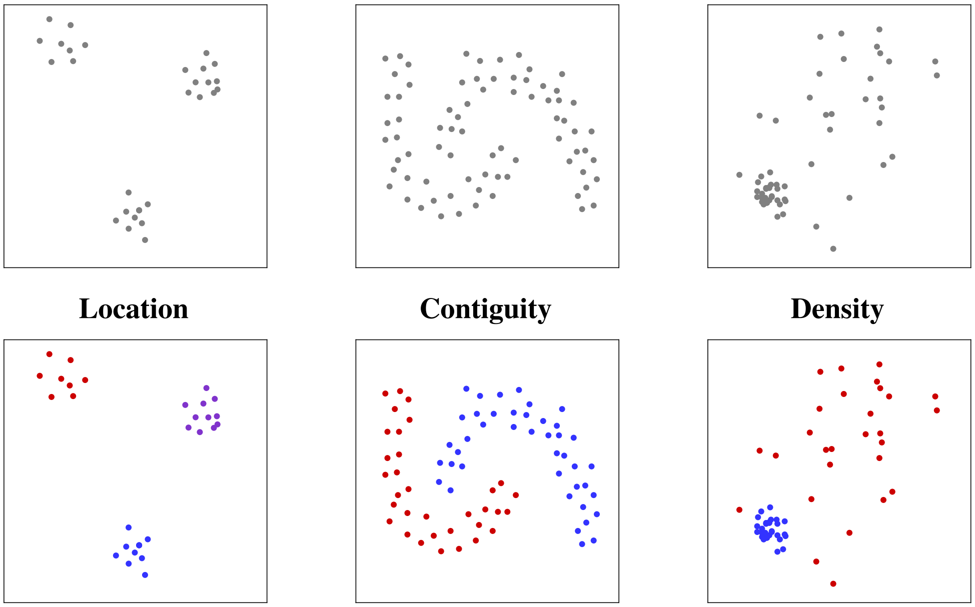


Figure 10.1: The concept of "grouping" data may mean different things in different settings. Groups may be defined, for example, by (a) their general location in the feature space, by (b) points that form shapes or contiguous sets, or (c) changes in the density of points in different parts of the feature space. Effective clustering methods depend on accurately quantifying the user's desired notion of similarity.

Vector Quantization and Data Compression

reorganize? move to end / applications? Initially, let us assume that "similarity" simply means that all the points in a cluster are close to some "exemplar" vector, in the sense of Euclidean distance. If so, any point in the cluster can be reasonably approximated by the exemplar; this idea is the basis of a data compression technique called *vector quantization*.

Using image compression as an example, the basic idea of vector quantization is to break the image up into a sequence of patches, represented as vectors of pixel intensities, and create a dictionary of patches (either specific to the image of interest, or from a population of similar images). This dictionary is called the *codebook*. Then, each patch within the image to be compressed can be approximated by one of the patches in the codebook, and instead of sending the values of the pixels in the patch, we simply send the discrete index of the approximate patch in the codebook.

Suppose our original image is made up of m patches of n pixels each, taking on 2^b grayscale values, and we make a dictionary with K exemplar patches. Then, communicating the dictionary requires $K \cdot n \cdot b$ bits, and sending the image given the dictionary requires only $m \cdot \log_2(K)$ bits, compared to $m \cdot n \cdot b$ for the original image. If $K \ll m$, or if the dictionary can be reused across images and $\log_2(K) \ll n \cdot b$, this can substantially reduce the size of representing the image.

An example of this is shown in Figure 10.2 using a 128×128 image in $b = 8$ bit grayscale. Figure 10.2(a)–(b) illustrates how small patches of the image (red squares) can be sufficiently similar to one another that a single dictionary entry could be used to replace them. Figure 10.2(c) shows a scatterplot the intensities of pairs of neighboring pixels (grey points) in the image, along with $K = 64$ dictionary entries. (Since most pixels are similar in color to their neighbor, we see a lot of data falling near the diagonal of the plot.) Finally, Figure 10.2(d) shows a reconstruction of the image after compression, using

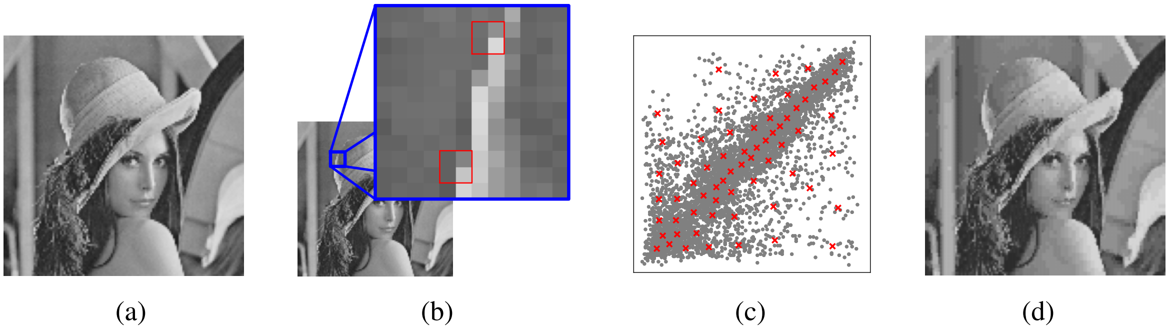

(a) (b) (c) (d)

Figure 10.2: Viewing the image (a) as a sequence of patches, (b) we see that many of these patches are similar to one another. (c) A scatter plot of neighboring pixel intensities found in the image; this is quantized by building a discrete dictionary of patches (red "x") that can stand in for any similar patches found in the image. (d) A reconstruction of the original image using 2×2 patches quantized into a dictonary of size 64, giving $\approx 5\times$ compression.

$K = 64$ patches of size $n = 2 \times 2$, reducing the representation size of the image by a factor of ≈ 5 (or, a *compression rate* of $\approx \frac{1}{5} = 0.2$).

Although vector quantization methods are no longer widely used for audio and video compression (mainly in very low-rate settings), it shows how understanding the underlying structure of the data and finding ways to group and summarize it is directly connected to efficiently representing and compressing the data. In fact, modern machine learning methods such as deep neural networks (Chapter **??** and latent space representations (Chapter 11) have been applied to yield high-quality image compression methods (see, e.g., **?**.

In the remainder of the chapter, we turn to the *how* of clustering, examining three common techniques for identifying clusters: K-means clustering (Section 10.1); probabilistic mixture models, particularly Gaussian mixture models (Section 10.2); and hierarchical agglomerative clustering (Section 10.3).

10.1 K-Means Clustering

Perhaps the simplest clustering algorithm is a technique called *k-means* clustering. In k-means, we have k clusters, each of which is described by a single point (the "cluster center") in the feature space of the data. Cluster membership is determined by a data point's similarity to this center point, as measured by standard Euclidean distance in the feature space[1]. Let us denote the k cluster centers by $\{\mu_1, \ldots, \mu_k\}$; each of the m data points $\{x^{(i)}\}$ is assigned to one of these clusters, and we denote the cluster to which data point i belongs as $z^{(i)}$.

The k-means algorithm is quite simple. We first initialize the values of the cluster centers μ; this can be done in a number of ways, which we examine in more detail in Section 10.1. Then, we iterate repeatedly between two steps:

Assignment For each data point i, set the cluster assignment $z^{(i)}$ to the cluster with the closest center,

$$\forall\, i : \qquad z^{(i)} = \arg\min_c \|x^{(i)} - \mu_c\| \tag{10.1}$$

(Break any ties arbitrarily but non-randomly.)

[1] With some effort, the algorithm can be generalized to use other similarity functions if desired.

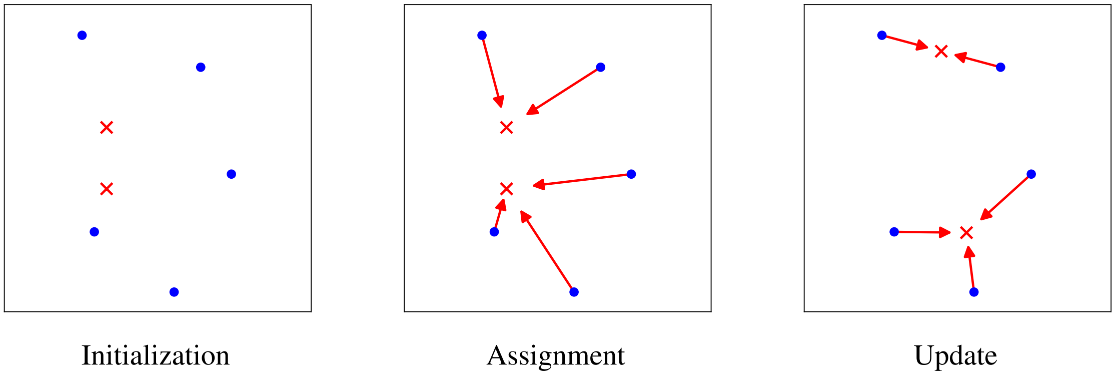


Figure 10.3: Basic steps of k-means. (a) Initialize the cluster centers (red); then iterate between (b) assigning each data point (blue) to its closest cluster center (indicated by arrows), and (c) updating the cluster center to the mean of the data currently assigned to that cluster.

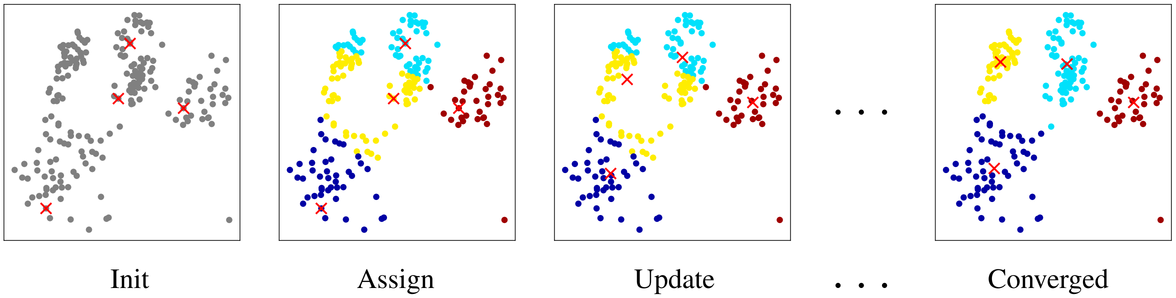


Figure 10.4: An example execution of k-means on a data set with $k = 4$. (a) Initialize the cluster centers. (b) Assign each data point to its nearest cluster (assignments shown via color). (c) Update the cluster center to the mean of its assigned data points. (d) Repeat these two steps until convergence.

Update For each cluster c, update the cluster center to be the mean of the assigned data (data with $z^{(i)} = c$).

$$\forall\, c: \qquad \mu_c = \frac{1}{m_c} \sum_{i:z^{(i)}=c} x^{(i)}, \qquad \text{where } m_c = |\{i : z^{(i)} = c\}| \tag{10.2}$$

These two steps are illustrated in Figure 10.3. We stop when the algorithm converges, i.e., the assignment step does not change the assignment of any data point. (This means, in turn, that the mean of the data assigned to each cluster will not change, and thus the algorithm will no longer update any $z^{(i)}$ or μ_c.) Figure 10.4 shows an execution of the algorithm on an example data set; after several iterations, the assignment and means cease to change and the algorithm is converged.

We can view the k-means procedure as a minimization of an objective function, specifically, sum of squared distances between the data and their assigned cluster:

$$J_k(z, \mu) = \sum_i \|x^{(i)} - \mu_{z^{(i)}}\|^2. \tag{10.3}$$

The two steps of k-means correspond to a coordinate descent procedure on this objective – keeping the μ_c fixed, each $z^{(i)}$ is involved with only one term of the sum, $\|x^{(i)} - \mu_{z^{(i)}}\|^2$. Clearly, the value of $z^{(i)}$

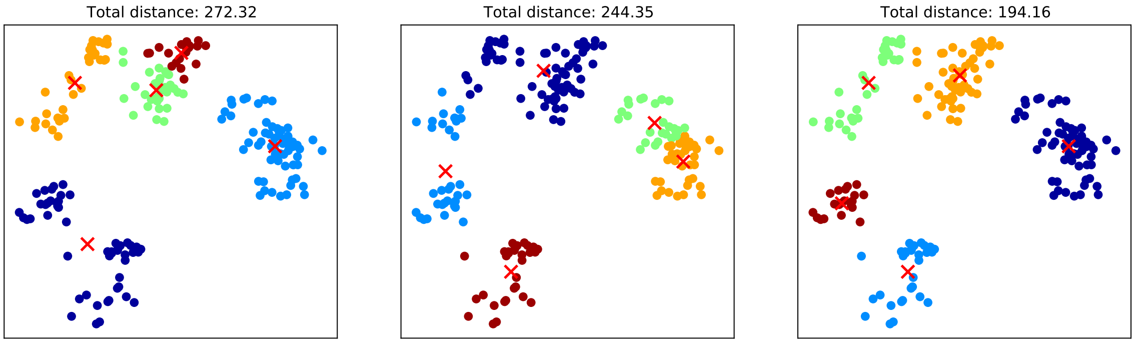


Figure 10.5: Different fixed points of k-means ($k = 5$) resulting from different initializations. We can compare the quality of a clustering solution by measuring its dissimilarity cost, e.g., Eq. (10.3) for k-means.

that minimizes this is the closest cluster c from Eq. (10.1). Similarly, if we keep the $\{z^{(i)}\}$ fixed, each cluster center μ_c can affect only those terms in the sum with $z^{(i)} = c$, so that

$$J_k(z, \mu) = \sum_{c=1}^{k} \sum_{i:z^{(i)}=c} \|x^{(i)} - \mu_c\|^2.$$

Then, the point μ_c that minimizes the total Euclidean distance from data $\{x^{(i)} : z^{(i)} = c\}$ is simply their centroid, or mean value.

Viewing k-means as a coordinate descent procedure is quite useful. For one thing, it allows us to easily see that k-means will always converge, since J is bounded below by zero, and must always strictly decrease in value until convergence. It also shows us that k-means is likely to have many possible local minima (stable assignments), since coordinate descent is fast but prone to local optima. Finally, it gives us a way to select between any such local minima we might find, by choosing the clustering with the lowest value of J. A common procedure to circumvent local optima is to simply run k-means multiple times, from different initializations, and keep track of the clustering that results in the lowest cost J at convergence. Figure 10.5 shows several examples of local optima found in this manner, with decreasing costs.

Initializing K-means

Given that the k-means algorithm can have many converged solutions, finding good initializations can be very important in practice. A few desirable properties usually guide the initializatoin process. In clustering, we are assuming inherently that the data form groups; we would therefore like to place the initial clusters close to these groups, and additionally to place them far away from each other. Finally, we would like the process to have sufficient randomness so that if we re-run the process, we can get significantly different inititalizations (and thus look for different solutions).

As discussed in Chapter 2.4, the curse of dimensionality implies that in high dimension (many features), if we sample μ from some simple, fixed distribution (Gaussian, uniform, etc.), it will be difficult to guarantee that μ is much closer to some data than others. A simple solution to this is to select some of the data points $x^{(i)}$ themselves as the cluster centers; assuming that the data do form clusters, this should place our centers μ_c within one of these clusters. A simple initialization rule is:

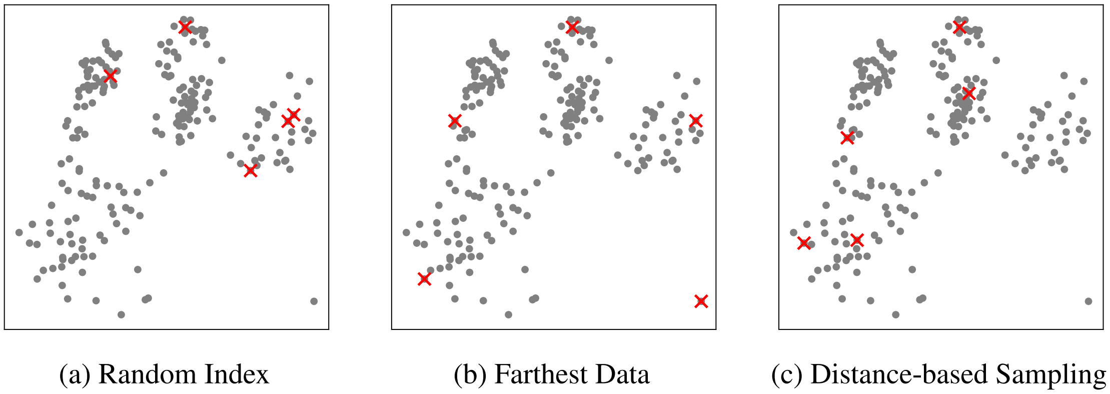

(a) Random Index (b) Farthest Data (c) Distance-based Sampling

Figure 10.6: Initializing k-means. (a) Selecting k random data points ensures that the cluster centers are near data, but may select points that are nearby (e.g., the leftmost clusters). (b) Selecting the farthest point from already-selected clusters ensures diversity, but may select outlier data that are not very representative (e.g., lower right cluster). Distance-based sampling combines the two concepts, giving points that are typically in dense regions of the space yet not too close to each other.

Random Index Select k data points uniformly at random (without replacement) from the data $\{x^{(i)}\}$ to be the cluster centers $\mu_1, \ldots, \mu_k$.

This initialization procedure is very simple and efficient, requiring only $O(k)$ work. However, this rule may select several centers within the same group of data. Figure 10.6(a) shows an example on two-dimensional data; using random index initialization, the procedure is happy to sample points that are arbitrarily close, for example the two nearby clusters on the right.

If we would like to ensure that the clusters are assigned to different data, we can encourage the initial cluster centers to be far apart. This leads to a slightly more complex procedure:

Farthest Select center μ_1 uniformly at random from $\{x^{(i)}\}$. Then, choose each μ_c to be the data point $x^{(i)}$ which is farthest from any already-chosen center:

$$\mu_c = \arg \max_{x \in \{x^{(i)}\}} \min_{c' < c} \|x - \mu_{c'}\|^2$$

Initialization then requires $O(m\,k\,n)$ work, since we must re-calculate the minimum distance for each data point after eack of our k cluster selections.

Unfortunately, this rule has a tendancy to choose *outlier* data, if present – a single data point that is very far from all the others will almost certainly be chosen as a cluster center, despite the fact that it is not near any *group* of data. Additionally, this illustrates that the initialization is not very random – after choosing the initial point, the method will often end up with effectively equivalent initializations, which makes it harder to find better local optima by re-running the algorithm. Figure 10.6(b) shows an example; the data point at the lower right will almost always be selected as one of the cluster centers using this approach.

To address these points, Arthur and Vassilvitskii [2007] proposed to initialize the cluster centers randomly, but using a distribution over the data weighted by distance to the nearest cluster, so that points that are far from the current centers will have higher probability, but we are more likely to sample a data point from a large *group* that is far from our current centers than to select a single outlier.

Distance-based sampling Select center μ_1 uniformly at random from $\{x^{(i)}\}$. Then, choose each μ_c sequentially according to the distribution, for example,

$$p(\mu_c = x^{(i)}) \propto \min_{c' < c} \|x^{(i)} - \mu_{c'}\|^2 \tag{10.4}$$

where $\propto$ indicates that we should normalize the distances to form probabilities that sum to one.

In terms of complexity, this approach is again $O(m\,k\,n)$. At a high level, this approach encourages selecting cluster centers that are distant from any already-selected points, but is also more likely to choose a point from among a large group (since any one of the points might be chosen). One such sample is shown in Figure 10.6(c); initial locations are mostly well-separated, while avoiding the outlier data point, and re-running the procedure will give a reasonably different initialization.

The specific sampling distribution in Eq. (10.4) is called "D^2 weighting" in Arthur and Vassilvitskii [2007], and its associated initialization procedure is called "k-means++". In addition to satisfying both high-level desiderata, Arthur and Vassilvitskii [2007] also prove that such initialization is (in expectation) not too far from the optimal clustering, i.e., that after initialization,

$$J^*_{opt} = \min_{\{\mu_c\}} \min_{\{z^{(i)}\}} \|x^{(i)} - \mu_{z^{(i)}}\|^2 \leq 8(\log k + 2)\, \mathbb{E}\Big[J(z,\mu)\Big]$$

where the expectation is over z and μ selected via D^2 weighting. Example initializations of each type are shown in Figure 10.6 on two-dimensional data.

Choosing the number of clusters k

Up to this point, we have assumed that the number of clusters, k, is fixed and known. However, it is often the case that we would like to discover this information from the data themselves; we do not know *a priori* how many groups exist in our dataset.

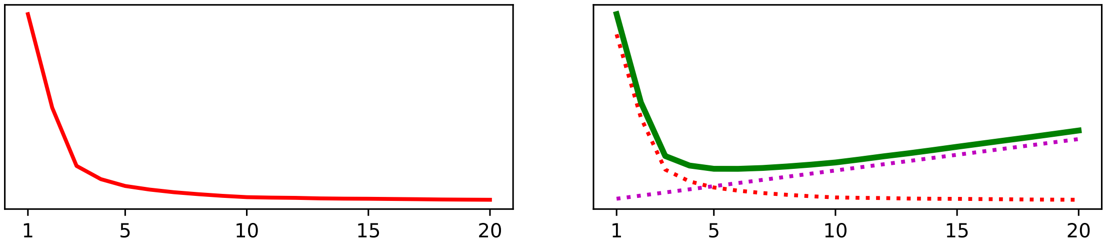
Unfortunately, we cannot use the loss function J_k in (10.3) to select the best value of k. Since after convergence, J_k measures the distance to the nearest cluster center, adding a cluster center will always decrease the optimal value of J_k. Consider adding an additional center after convergence – now, some data point will be closer to the new center, and so its cost will decrease, while any data that remain closer to their old center will have their cost unchanged. Figure 10.7 shows the cost J_k as a function of the number of clusters k for the same data as Figure 10.5 (after optimizing and using 5 random restarts).

Selecting k is thus a *model selection* or *model complexity* problem – increasing k increases the representational power of our clustering, and thus always appears to fit the data better. Moreover, we cannot simply use hold-out data with the loss J, since we would need to assign each data point to a cluster (say, choosing the nearest), and end up with the same problem.

Instead, we can apply a *penalized loss* framework to determine k, in which we add a complexity penalty to our cost function that increases with the number of clusters. Then, as we increase k, we will better fit the data, but the complexity cost will increase. If the additional cluster does not improve the data fit enough to compensate for the increase in complexity, we can reject it in favor of the simpler model.

A common choice is to use a penalty based on the "Bayesian information criterion" or BIC penalty for probabilistic models, exploiting a connection between k-means and Gaussian mixture models (discussed in the next section). For the k-means problem, a simplified version of this procedure gives the complexity-penalized loss,

$$\tilde{J}_k(z,\mu) = \Big[\frac{1}{m}\sum_i \|x^{(i)} - \mu_{z^{(i)}}\|^2\Big] + k \cdot n \cdot \frac{\log m}{m}.$$



(a) (b)

Figure 10.7: Model selection for k-means. (a) Increasing the value of k will always decrease the dissimilarity cost J_k, since more clusters mean that each data point can be nearer to a cluster. (b) We can penalize for more complex models by adding a penalty for increasing k, such as BIC; the sum of these two terms, $\tilde{J}_k$, can then increase with k if adding more clusters does not reduce the distances J_k enough to compensate for the increased complexity. **add labels**

At each value of k, we compute $J_k(z, \mu) + R(k)$, and select k by the smallest total loss. As J becomes small, the sum will eventually be dominated by R; see Figure 10.7(b). A more in-depth exploration of model selection to determine k for k-means can be found in Pelleg and Moore [2000].

10.2 Mixture Models

Another useful paradigm for identifying groups among data is the concept of *mixture models*. Mixture models are probability distributions that explicitly incorporate the concept of a discrete collection of groups, where we associate each group (or cluster) z with its own probability distribution $p_z(x)$ (called the *mixture component*). We define a generative probability distribution by first selecting from which cluster z our data point originates, then drawing x from mixture component $p(x|z) = p_z(x)$.

As with k-means, let us begin by assuming that we have a known number of clusters k. Let $p(z)$ be a categorial distribution with parameter π, where $\pi_z = p(z)$ is the probability of selecting cluster z. The distribution of an observation x is then the marginal,

$$p(x) = \sum_z p(z)\, p(x|z) = \sum_z \pi_z\, p_z(x),$$

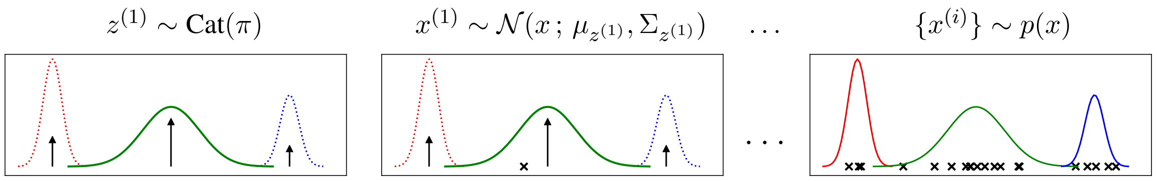
a weighted average of the mixture components $p_z(x)$. In practice, we will observe some data $D = \{x^{(i)}\}$, and attempt to discover the clusters by fitting our mixture model $p(x)$ to the data. As we shall see, the fact that this model provides a generative probability distribution for the data also gives a number of advantages, such as the ability to simulate examples from the model by sampling, or imputing the value of missing features in our data. **discuss more later?**

Gaussian Mixture Models

Perhaps the most widely used type of mixture model is the *Gaussian mixture model*, sometimes abbreviated as GMM. A Gaussian mixture model associates a Gaussian distribution with each group (cluster), which can then be characterized by the Gaussian parameters – its mean and covariance – which describe the center and the amount and direction of spread for the data points in that cluster.

Example 10-1 : Univariate Gaussian Mixture

Consider a distribution over a single scalar feature x, consisting of three Gaussian components. The sampling process for this distribution – selecting a component $z^{(i)} \in \{1, 2, 3\}$ according to the probabilities π, then drawing $x^{(i)}$ from that component's Gaussian, result in a collection of samples that form groups defined by the locations (means) and spread (variance) of the three components:



$z^{(1)} \sim$ Cat(π) $\quad$ $x^{(1)} \sim \mathcal{N}(x\,;\,\mu_{z^{(1)}}, \Sigma_{z^{(1)}})$ $\quad \ldots \quad$ $\{x^{(i)}\} \sim p(x)$

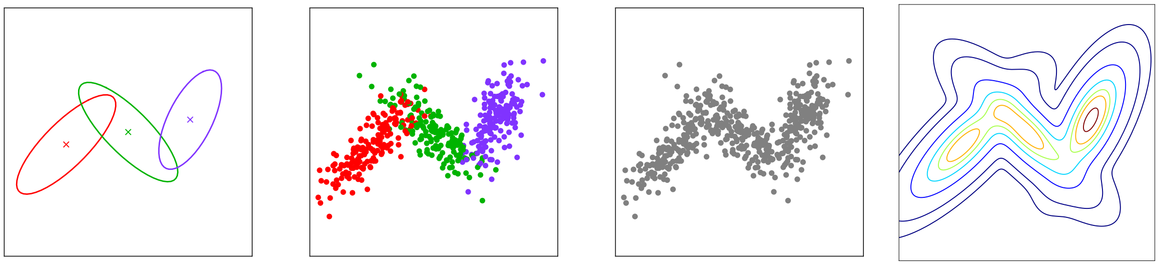
Our sampling process actually generates samples of pairs, $\{(z^{(i)}, x^{(i)})\} \sim p(Z, X)$, including both the component ID $z^{(i)}$ and the real-valued feature $x^{(i)}$ of each data point i. We call these joint observations the *complete data*. If we were able to observe the complete data, it would simple to recover an estimate of the mixture components; as when learning a Gaussian Bayes classifier (Chapter 1), the maximum likelihood estimates are given by the empirical estimates,

$$\hat{\pi}_z = \frac{m_z}{m} = \frac{\#\{z^{(i)} = z\}}{m} \qquad \hat{\mu}_z = \frac{1}{m_z} \sum_i x^{(i)} \qquad \hat{\Sigma}_z = \frac{1}{m_z} \sum_i (x^{(i)} - \hat{\mu}_z)^T \odot (x^{(i)} - \hat{\mu}_z)$$

where $\odot$ is the outer product of the two vectors. Unfortunatly, in practice we will not observe Z, and will thus need to work from the *incomplete data*, $\{x^{(i)}\} \sim p(X)$, in which the cluster ID is hidden from us. Models that have variables, like Z, that are used to explain patterns in the data but never observed directly are sometimes called *latent variable models*.

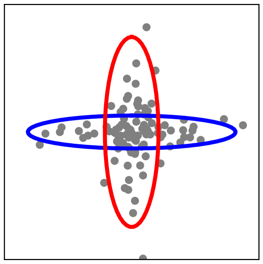
Example 10-2 : 2D Gaussian Mixture Model

Consider a two-dimensional Gaussian mixture distribution, again with three components. Each component is defined by a mean vector and covariance matrix, indicated in two dimensions by an ellipsoidal shape. If we know which cluster each data point came from (indicated by the color), it is easy to recover the location and shape of each cluster. In practice, however, we must work from the incomplete data (gray points), whose distribution matches the marginal $p(X)$.



Components $\{\mu_c, \Sigma_c\}$ Complete data $\{(z^{(i)}, x^{(i)})\}$ Incomplete data $\{x^{(i)}\}$ Marginal $p(X)$

In mixture model clustering, the form of the mixture component defines what it is that makes the data in a cluster "similar", and able to form a group. So for example, in a GMM, a cluster is defined by an ellipsoidal distribution, i.e., a small "blob" of data. This means that our model can differentiate between groups of data that differ in either their mean (as in k-means) and/or in the direction of their spread; the figure at right shows an example of the latter, in which the two groups of data have similar means, but one extends vertically while the other extends horizontally.



Notice that, if we were given the component distributions $p_z(X)$, it would not be too difficult to guess which data were associated with each cluster. Conversely, if we were given the associations (the latent $z^{(i)}$), we could easily estimate $p_z(X)$ for each cluster z. This observation suggests an iterative estimation procedure, similar to k-means, in which we alternate between estimating the Z values and the component distributions, called *Expectation-Maximization*, or EM. We first describe the EM algorithm procedurally, specialized to the case of Gaussian mixture components, then derive it more formally using a more general setting.

Expectation-Maximization for GMMs

The Expectation-Maximization, or EM, algorithm for fitting Gaussian mixture models can be viewed as a generalization of k-means. Taking the total number of clusters k to be fixed, we first initialize our model somehow, selecting initial parameters $\hat{\pi}$ and $\hat{\mu}_z$, $\hat{\Sigma}_z$ for each cluster z in a manner discussed in the sequel. Then, EM operates iteratively, by alternating between two steps:

E-step: For each data point i, compute the *responsibility* $r_z^{(i)}$ of each cluster z for generating that data point:

$$\forall\, i, z: \qquad r_z^{(i)} = p(Z = z | X = x^{(i)}) = \frac{p(Z = z, X = x^{(i)})}{p(X = x^{(i)})} = \frac{\pi_z \mathcal{N}(x^{(i)}\,;\, \mu_z, \Sigma_z)}{\sum_c \pi_c \mathcal{N}(x^{(i)}\,;\, \mu_c, \Sigma_c)}$$

By definition, the responsibilities sum to one: $\sum_z r_z^{(i)} = 1$.

M-step: For each cluster z, update the parameters using a weighted average with weights $r_z^{(i)}$:

$$\hat{\pi}_z = \frac{\hat{m}_z}{m} = \frac{\sum_i r_z^{(i)}}{m} \qquad \hat{\mu}_z = \frac{1}{\hat{m}_z} \sum_i r_z^{(i)}\, x^{(i)} \qquad \hat{\Sigma}_z = \frac{1}{\hat{m}_z} \sum_i r_z^{(i)}\, (x^{(i)} - \hat{\mu}_z)^T \odot (x^{(i)} - \hat{\mu}_z)$$

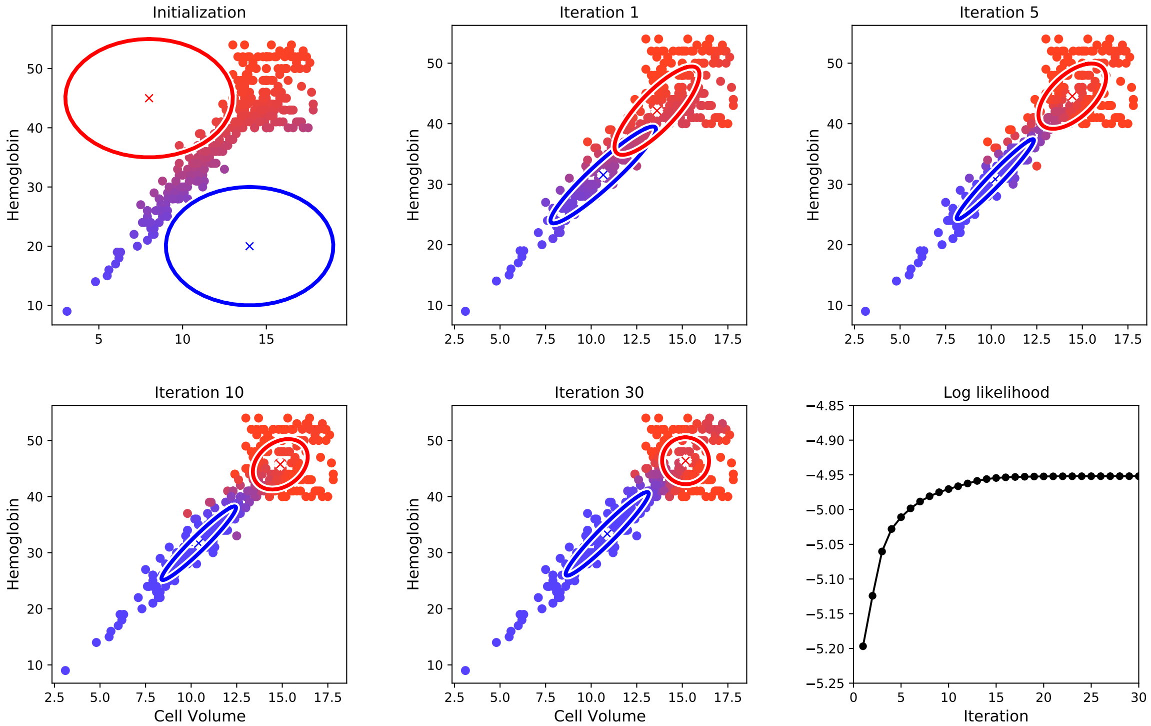
Intuitively, the EM algorithm alternates between finding a “soft” assignment of each data point to the clusters (the responsibility), and estimating the parameters of the cluster component using this assignment. If a given data point $x^{(i)}$ is far more likely to have come from some cluster z than the others, the responsibility will be high for that cluster, i.e., $r_z^{(i)} \approx 1$, and low for the others. If a data point is ambiguous, and could have come from any of several clusters, its influence will be split among the clusters it might have come from. The weighted averages then evaluate the empirical averages, accounting for this potential splitting of each data point among several clusters.

We show in the next section that this update procedure iteratively maximizes the likelihood, or equivalently minimizes the average negative log-likelihood, of the (incomplete) data under the model, i.e., it minimizes the loss,

$$J_{\text{NLL}}(k, \pi, \mu, \Sigma) = -\frac{1}{m} \sum_{i=1}^{k} \log \Big[\sum_z \pi_z\, \mathcal{N}(x^{(i)}\,;\, \mu_z, \Sigma_z) \Big].$$

Example 10-3 : GMM clustering

Consider a data set of hospital patients, some of whom have chronic kidney disease [Soundarapandian et al., 2015]. We examine two features of each patient: the packed cell volume of their blood, X_1, and their measured hemoglobin levels, X_2. In general, decreased cell volume and hemoglobin are indicative of anemia. We fit a two-cluster model to the data; each plot shows the current value of the component parameters (red and blue ovals), and the current estimated responsibility of each data point (red if $r_1^{(i)} = 1$, blue if $r_1^{(i)} = 0$, and a blend for values in between).



After initialization, the red cluster has higher responsibility to the data at the top of the plot, and the blue cluster to the data nearer the bottom. After performing the M-step, the means and covariances are updated to match these (weighted) data, and the red cluster becomes relatively more responsible for the data at the top right, and so on, until the red and blue clusters are capturing the location and shape of the "normal" patients (red) and "anemic" patients (blue). The average log-likelihood of the model at each iteration is also shown; early updates increase the log-likelihood significantly, after which the model begins to converge.

It is easy to see that the k-means procedure is a special case of EM. Assume fixed values for both the weights, $\pi_z = \frac{1}{k}$, and the covariances, $\Sigma_z = \epsilon I$ (variance ϵ in each feature), and select ϵ to be very small. Then, the responsibility $r_z^{(i)}$ will be highest for whichever cluster has the closest mean, and as $\epsilon \to 0$, its value will approach one, while the others approach zero. The M-step update for $\hat{\mu}_z$ then exactly corresponds to the cluster center update in k-means.

Deriving Expectation-Maximization

Like k-means, EM can be framed as a form of coordinate descent on its objective function, the negative log-likelihood of the data. However, the "coordinates" being optimized over at each step correspond to (parameters of) distributions.

We begin by re-writing the negative log-likelihood of the data D as a KL divergence between the empirical distribution, $\hat{p}(X) = \frac{1}{m}\sum_i \delta(X - x^{(i)})$, and our mixture model $p(X\,;\,\theta) = \sum_z p(z\,;\,\theta)p(X|z\,;\,\theta)$, with parameters θ – for example, $\theta = (\pi, \{\mu_z\}, \{\Sigma_z\})$ for the Gaussian mixture model. Then,

$$\begin{aligned} D\big(\hat{p}(X) \| p(X\,;\,\theta)\big) &= \mathbb{E}_{\hat{p}}\big[\log \hat{p}(X) - \log p(X;\theta)\big] \\ &= -\mathrm{H}[\hat{p}] - \frac{1}{m}\sum_i \log p(X = x^{(i)}\,;\,\theta) \end{aligned}$$

The empirical entropy $\mathrm{H}[\hat{p}]$ depends only on the (observed) data, and is thus a constant; so minimizing the term on the right – the negative log-likelihood of the data – is equivalent to minimizing the divergence $D(\,\hat{p}(X)\|p(X;\theta)\,)$.

Now, let us relate this quantity to an objective function involving the complete distribution, $p(X, Z\,;\,\theta)$. We introduce an arbitrary distribution $q(Z|X)$, and apply Jensen's inequality:

$$\begin{aligned} D\big(\hat{p}(X)||p(X;\theta)\big) &= -\mathrm{H}[\hat{p}] - \mathbb{E}_{\hat{p}}\big[\log p(X;\theta)\big] \\ &= -\mathrm{H}[\hat{p}] - \mathbb{E}_{\hat{p}}\big[\log \sum_z p(X,z;\theta)\big] \\ &= -\mathrm{H}[\hat{p}] - \mathbb{E}_{\hat{p}}\Big[\log \sum_z q(z|X)\frac{p(X,z;\theta)}{q(z|X)}\Big] \\ &\le -\mathrm{H}[\hat{p}] - \mathbb{E}_{\hat{p}}\mathbb{E}_{q(Z|X)}\Big[\log \frac{p(X,Z;\theta)}{q(Z|X)}\Big] \\ &= -\mathrm{H}[\hat{p}\,q(Z|X)] - \mathbb{E}_{\hat{p}(X)q(Z|X)}\big[\log p(X,Z;\theta)\big] \\ &= D\big(\hat{p}(X)\,q(Z|X)||p(X,Z;\theta)\big) \end{aligned}$$

In other words, our desired negative log-likelihood (divergence) objective is bounded above by a divergence of the complete model over X and Z, which involves the empirical distribution (data) over X and an arbitrary "extension" of this distribution to the unobserved variable Z.

Then, EM is simply a joint minimization of this upper bound,

$$\min_\theta \min_q D\big(\hat{p}(X)q(Z|X)||p(X,Z;\theta)\big),$$

alternating between optimizing over the distribution $q(Z|X)$, then over the parameters θ:

E-step: Minimize over $q(Z|X)$. We are only interested in the values of $q(Z|X)$ at the data points, $x^{(i)}$; since Z is discrete (the cluster label), we can represent each $q(Z|X = x^{(i)})$ as a vector of "responsibilities", $q(Z|X = x^{(i)}) = [r_1^{(i)}, \ldots, r_k^{(i)}]$, which sum to one.

To update $q(Z|X)$, we can re-write the divergence as,

$$\begin{aligned} D\big(\hat{p}(X)q(Z|X)\|p(X,Z;\theta)\big) &= D\big(\hat{p}(X)\|p(X;\theta)\big) + \mathbb{E}_{\hat{p}}\mathbb{E}_q\Big[\log \frac{q(Z|X)}{p(Z|X;\theta)}\Big] \\ &= D\big(\hat{p}(X)\|p(X;\theta)\big) + D\big(q(Z|X)\|p(Z|X;\theta)\big) \end{aligned}$$

and see that the minimum over q is given by selecting $q(Z|X) = p(Z|X;\theta)$, the conditional distribution of Z defined by the current model. It will also be useful to note that, at this value of q, the second divergence term is zero, and so our two objectives (the complete distribution divergence and the marginal divergence) are equal.

M-step: Minimize[2] over the model parameters θ. Since this involves the complete log-likelihood, it is often easy:

$$D\big(\hat{p}(X)q(Z|X)||p(X,Z;\theta)\big) = -\mathrm{H}[\hat{p}(X)\,q(Z|X)] - \mathbb{E}_{\hat{p}}\mathbb{E}_q\big[\log p(X,Z;\theta)\big]$$

[2] The "maximization" in EM refers to maximizing the (expected complete) log-likelihood; this is equivalent to minimizing its negative value, as in our derivation.

where the first (entropy) term does not depend on θ. The second (expectation) term is sometimes called the **expected complete log-likelihood**; typically, if it is easy to optimize the model using the complete data $\{(z^{(i)}, x^{(i)})\}$, it will also be easy to optimize the expected complete log-likelihood. For any mixture model, $p(z, x) = \pi_z \, p_z(x)$, the index z indicates which mixture component $p_z(x)$ to use. Then, the expected complete log likelihood is,

$$\begin{aligned}\mathbb{E}_{\hat{p}}\mathbb{E}_q\big[\log p(X, Z; \theta)\big] &= \mathbb{E}_{\hat{p}}\mathbb{E}_q\Big[\sum_z \mathbb{1}[Z = z] \log \pi_z \;+\; \sum_z \mathbb{1}[Z = z] \log p_z(X)\Big] \\ &= \frac{1}{m}\sum_i \Big[\sum_z q(Z = z|x^{(i)}) \log \pi_z \;+\; \sum_z q(Z = z|x^{(i)}) \log p_z(x^{(i)})\Big].\end{aligned}$$

The two terms can be optimized independently, since they have no parameters in common. The maximum likelihood estimate of π is simply $\hat{\pi}_z = \frac{1}{m}\sum_i q(Z = z|x^{(i)})$, identical to the estimate in the Gaussian case. The term involving any given component $p_z(\cdot)$, meanwhile, is simply a maximum likelihood objective using a *weighted* data set, with weights $q(Z = z|x^{(i)})$ on each data point i. Thus, assuming the components also do not share parameters, fitting them requires only a minor variation on a standard maximum likelihood estimator for whatever type of model we have chosen for $p_z(x)$.

In the Gaussian case, for example, our standard maximum likelihood estimator of them mean using the complete data becomes,

$$\hat{\mu}_z = \frac{1}{\hat{m}_z}\sum_i \mathbb{1}[z^{(i)} = z]\, x^{(i)} \qquad \Longrightarrow \qquad \hat{\mu}_z = \frac{1}{\hat{m}_z}\sum_i q(Z = z|x^{(i)})\, x^{(i)},$$

(Complete data) (Expected Complete LL)

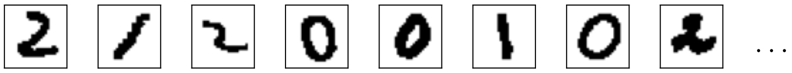
i.e., we go from the empirical average over the data originating from cluster z, to a weighted average using each data point's probability of coming from cluster z.

A few properties of these updates are useful to note. First, we are performing coordinate descent on our bound of the negative log-likelihood, so the bound cannot increase at each iteration, and (assuming the log-likelihood is bounded) will converge to a local optimum. Second, after each E-step, the bound matches the negative log-likelihood of $p(X; \theta)$; hence this must also be a local optimum of the desired (marginal) likelihood function. Thus, under fairly weak conditions this process converges to a local optimum of the (generally non-convex) negative log-likelihood of the mixture model $p(X; \theta)$.

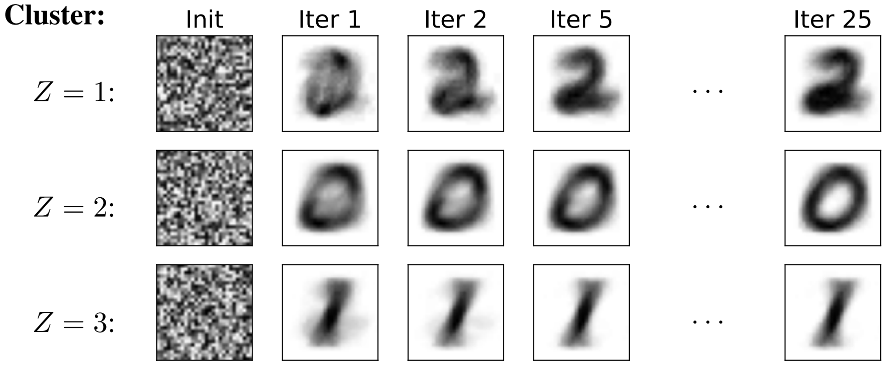
In general, EM can be applied to any model with unobserved variables, including discrete latent variables (mixture models), continuous latent variables (latent space methods; see Chapter 11), or (with some care) missing features in our data [**?**]. Many generalizations and alternate versions of EM have also been developed, including "Majorize-Maximize" algorithms, which may use a different identity to bound ("majorize") the objective function [**?**]; methods which use gradient-based or incomplete optimization over θ and q [**?**], or which use a sample from $q(Z|X = x^{(i)})$ (called **stochastic EM**) or which use its most likely value (called **hard EM**).

Example 10-4 : Clustering handwritten digits

To illustrate a simple non-Gaussian mixture model, consider modeling a dataset of handwritten digit images derived from MNIST [LeCun and Cortes, 2010]:



Each image is 28×28 binary pixels and contains one of the digits $\{0, 1, 2\}$. We fit our training data using a trivial component model $p_z(x)$ with one probability p_z^{ij} per pixel ij, indicating the (independent) probability that that pixel is "on" in the image x. (Then, the maximum likelihood estimates of the p_z^{ij} are simply the weighted averages of the images.) A few iterations of EM with $K = 3$ quickly finds reasonable clusters:



Selecting the number of clusters

Mixture models provide a fully defined probability distribution over X. One significant resulting advantage is that model complexity issues, such as selecting the number of clusters k, can be resolved using test likelihood on a held-out data set, or using cross-validation.

If we have too few data to allocate a reasonable hold-out set, or wish to use all our data for fitting the model for some other reason, we can also use a penalized likelihood method, in which we optimize a sum of two quantities: the (negative) log-likelihood, which measures how well our model fits the training data, plus an additional cost that increases with the complexity of the model. Many such penalties are available for probability models; for example, the **Bayesian Information Criterion**, or BIC, penalty, penalizes based on the number of parameters in the model, R:

$$J_{\text{BIC}}(\theta) = -\frac{1}{m} \sum_i \log p(x^{(i)}; \theta) \;+\; \frac{R}{2m} \log m$$

For a fixed complexity model (value of R), the optimal θ is just the maximum likelihood estimate. But, when comparing between two models with different numbers of parameters, the BIC penalty compares whether the improvement in fit is sufficient to compensate for the increase in R. (Another, related complexity penalty is the **Aikike Information Criterion**, or AIC, which penalizes by $\frac{R}{m}$ instead. In general, using AIC will select more complex models than BIC, since usually $\log m > 2$.)

To see how this BIC penalized model selection relates to k-means, let us assume fixed values for both the cluster probabilities $\pi_z = \frac{1}{k}$ and covariances $\Sigma_z = I$ (unit variance in all features). Then, the only parameters are the k cluster centers, so $R = k \cdot n$. Approximating the log-likelihood using the "hard assignment" to $z^{(i)}$ gives: $\log p(x^{(i)} | z^{(i)}) = -\frac{1}{2} \|x^{(i)} - \hat{\mu}_{z^{(i)}}\|^2$. This leaves us with the complexity-penalized loss $\tilde{J}_k$ from Section 10.1. For a more sophisticated treatment, for example using a maximum likelihood covariance estimate, see Pelleg and Moore [2000].

In practice, held-out data is often more forgiving of complex models that do not produce severe ovefitting, while penalized likelihood methods like BIC penalize the model for increased complexity (number of parameters) no matter how those parameters are used. As an example, Figure 10.8 compares the BIC-penalized training loss versus the negative log-likelihood of a held-out test data set. BIC finds its minimum at $k = 3$ clusters (arguably the "right" number, given that the data include only three digits); however, hold-out loss indicates that even with higher values of k the model has still not overfit.

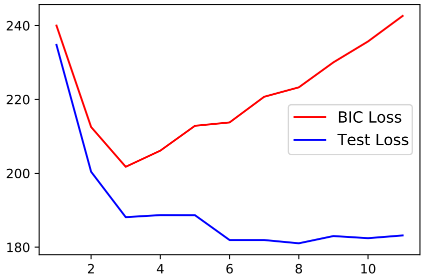


Figure 10.8: Comparing model selection with BIC to hold-out loss on clustering handwritten digits.

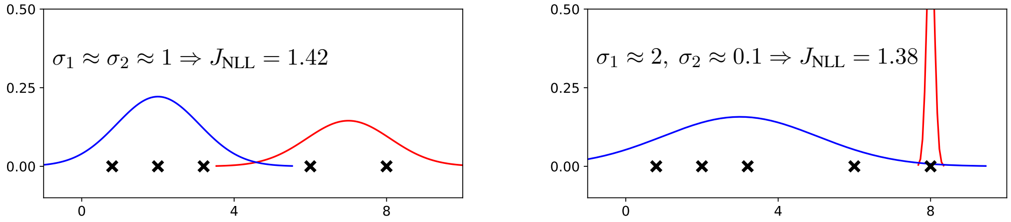
Local optima and singularities

The EM procedure is guaranteed to find a local optimum of the log-likelihood of our data. However, in general the log-likelihood of our models is not a convex function of their parameters, and so we may find any of a number of local optima. As with other non-convex optimization problems, we may re-run the procedure to try to identify different optima, and use the objective function to select among them.

However, an additional complexity arises with many mixture models over continuous-valued x. When a mixture component is assigned only a small number of data, it may be able to overfit, leading to an infinite training log-liklelihood. This effect is called a *singularity* in the probability model.

Example 10-5 : Singularities in Gaussian mixture models

Consider a two-cluster Gaussian mixture model on five univariate data points. There are many local optima of the log-likelihood; two settings near local optima are:



We can see that in the second case, one cluster is explaining only a single data point, which allows it to give that point infinite probability as $\sigma_2 \to 0$; as we continue to optimize, we find that $J_{\text{NLL}} \to -\infty$ on the right. This is not a useful solution for clustering, but in fact the global optima of the GMM model are actually at these limit points.

The issue is that many continuous probability density models are unbounded, and if a component is fit using maximum likelihood on too few data, we can find such a singularity in the log-likelihood. In practice, these issues are not difficult to overcome, however – we can either ignore such optima and search for the best non-singular optimum, or bound our parameters away from singular settings, for example by regularizing our estimates of the variance (most commonly, by adding a small constant to the diagonal entries). From the Bayesian statistical perspective, this corresponds to placing a prior on the covariances, and using the maximum *a posteriori* estimate of the parameters rather than maximum likelihood.

10.3 Heirarchical Agglomerative Clustering

Not all clustering algorithms fit the framework of optimizing an objective function with cluster assignments. In this section, we examine a widely used clustering algorithm that is defined in a more procedural way, and provides an entire hierarchy of clusters, rather than a single grouping.

Agglomerative clustering works by repeatedly merging clusters that are deemed similar to form a larger cluster. Suppose that we have a measure of dissimilarity D between data points, for example, simple Euclidean distance: $D(x, x') = \|x - x'\|$. We first extend this measure of dissimilarity to measure the dissimilarity between *sets* of data points, $D(C_1, C_2)$. This set-based dissimilarity is called the *linkage* function; we examine several ways of performing this extension in Section 10.3.

Then, given the linkage function, we can proceed to form clusters from the data by agglomeration. First, we make every data point $x^{(i)}$ its own cluster, so that we have m clusters total. Then, we find the two most similar clusters, and merge them together to form a single cluster. We repeat this process, at each step identifying the most similar pair of clusters and merging them, until all data points have been merged into a single cluster, or until some stopping condition (a fixed number of clusters k, or when all clusters are more dissimilar than some threshold). This procedure is illustrated in Figure 10.9.

We can visualize the sequence of merges using a diagram called a *dendrogram* (see Figure 10.9). The dendrogram shows the sequence of merges as a tree, in which the leaves are the individual data points, and the structure indicates in which order the data were merged. For each merge, we can connect the two clusters with a line whose height indicates those clusters' dissimilarity. Assuming that the linkage function satisfies a basic monotonicity property that $D(C_1, C_2 \cup C_3) \geq \min[D(C_1, C_2), D(C_1, C_3)]$, each merge will have higher dissimilarity that those before it, and the agglomeration process will grow the dendrogram upward.

One advantage of the dendrogram is that we can easily see the impact of stopping the clustering at different numbers of clusters. A horizontal slice shows how many clusters would be created if we stopped at a given dissimilarity. Similarly, large vertical steps indicate merging clusters that are not very similar to one another, or equivalently a wide number of dissimilarity thresholds that would result in the same clustering. This can often suggest the correct number of clusters. For example, in Figure 10.9 we see large jumps in dissimilarity when we merge from three to two clusters, or two to one, indicating that the data probably form at least three clusters.

Computational complexity

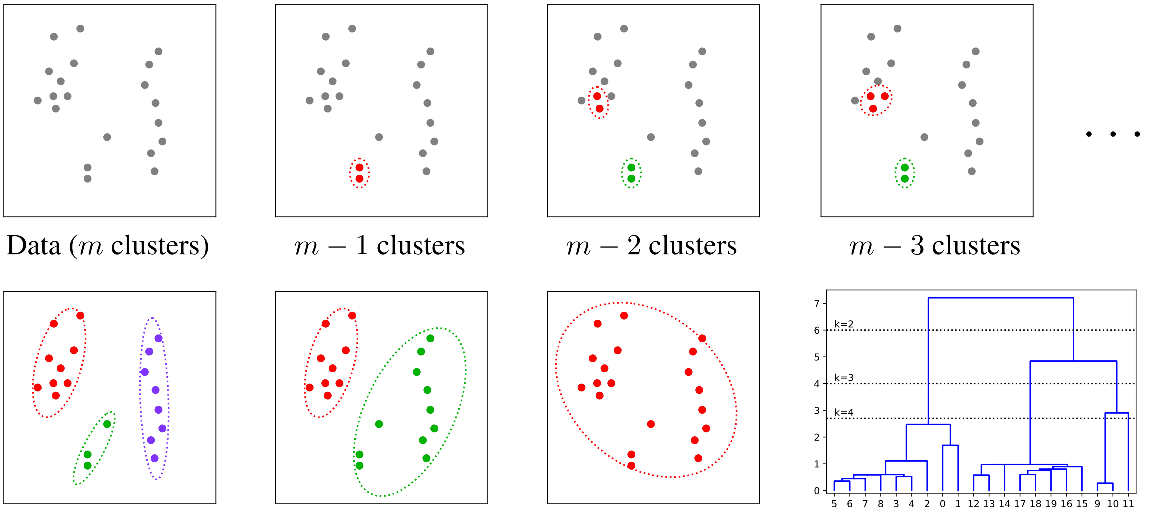
One reason for the popularity of agglomerative clustering is that the procedure is reasonably efficient. Given m data points in n dimensions, let us assume that the cost of computing the distance between any two clusters is $O(n)$ (we shall return to this point shortly). Then, we can calculate the computational complexity of the full procedure.

At the first step of the algorithm, we compute all distances between each pair of data points, at a cost of $O(n\, m^2)$. We then organize them in a heap or other structure to easily access the smallest distance, with cost $O(m^2 \log(m^2)) = O(m^2 \log m)$.

During the iterative portion of the algorithm, each step selects the smallest pair, merges them, and then updates the distances of the resulting new cluster with all existing clusters. Each such update takes $O(n + \log m)$ operations (to compute, then remove and insert the distance into the heap), and for the first iteration, there are $(m-1)$ other clusters to recompute. For the second iteration, there are $(m-2)$, and so on, until we have only two clusters and one distance. Thus, the total for all iterations is

$$(m-1)O(n + \log m) + (m-2)O(n + \log m) + \ldots \quad = O(n\, m^2 + m^2 \log m),$$

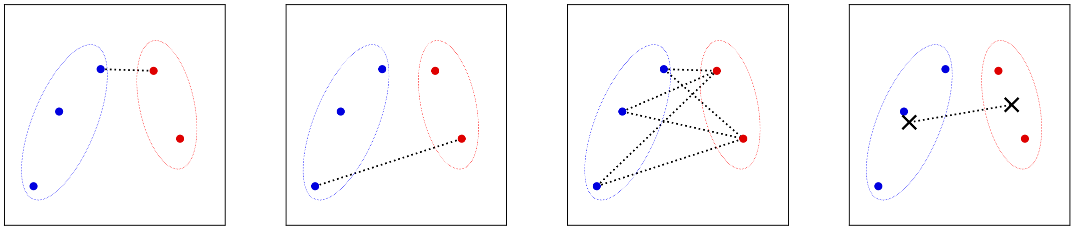
the same computational complexity as the initial sorted distance computations.



Data (m clusters) $m-1$ clusters $m-2$ clusters $m-3$ clusters

3 clusters 2 clusters 1 cluster Dendrogram

Figure 10.9: Agglomerative clustering procedure. Initially, each point is assigned to its own cluster; then, at each step, the most similar pair of current clusters is merged to form a larger cluster. The process can be stopped at any time, and the entire sequence of merge operations can be saved and displayed as a dendrogram.



Single Linkage Complete Linkage Average Linkage Centroid Linkage

Figure 10.10: Several different ways to define cluster dissimilarity in terms of individual data dissimilarity (here, Euclidean distance). (a) “Single linkage” uses the minimum distance between any pair of points in the two clusters; (b) “complete linkage” uses the maximum distance; (c) “average linkage” uses the average of all pairwise distances; and (d) “centroid linkage” uses the distance between the centroids of the data in each cluster.

Thus, the total complexity of the entire algorithm is $O(n\,m^2 + m^2 \log m)$. In a practical sense, both n and $\log m$ are usually quite small compared to m^2, so the running time of the clustering procedure is approximately quadratic.

Linkage functions

Let us now consider how to compute dissimilarity between pairs of clusters. Assume that we have a dissimilarity score between data points $D(x, x')$; for the purposes of visualization, we will assume Euclidean distance. Given two sets of points S_1, S_2, we can extend $D(\cdot,\cdot)$ to evaluate their dissimilarity in a number of ways; the most common choices include:

Single linkage. Choose the minimum distance between any pair of data points in S_1 and S_2, i.e.,

$$D_{\min}(S_1, S_2) = \min_{x \in S_1, x' \in S_2} D(x, x')$$

This definition implies that two clusters are similar if they are near each other at any point.

Complete linkage. Choose the largest (maximum) distance between any pair of points,

$$D_{\max}(S_1, S_2) = \max_{x \in S_1, x' \in S_2} D(x, x')$$

Complete linkage corresponds to an assumption that two clusters are similar *only* if all data points in the clusters are similar.

Average linkage. Choose the average of the dissimilarities between members of S_1 and S_2,

$$D_{\text{avg}}(S_1, S_2) = \frac{1}{|S_1|\,|S_2|} \sum_{x \in S_1} \sum_{x' \in S_2} D(x, x')$$

Centroid linkage. Evaluate the pointwise dissimilarity between the centroids (means) of each cluster, i.e., two clusters are similar if their centroids are close:

$$D_{\text{cen}}(S_1, S_2) = D\Big(\frac{1}{|S_1|} \sum_{x \in S_1} x \;,\; \frac{1}{|S_2|} \sum_{x' \in S_2} x' \Big)$$

Visualizations of each of these possibile linkage choices are shown in Figure 10.10.

Recall that, in Section 10.3, we assumed that computing the cluster dissimilarities required only $O(n)$ work. Although our linkage choices *appear* to involve all pairs of data, in fact it is easy to maintain sufficient statistics of each cluster during agglomeration which allow $O(n)$ or $O(1)$ evaluations. Suppose that we have already evaluated $D(S_1, S_2)$ and $D(S_1, S_3)$, and now wish to merge the clusters S_2,S_3 and recompute distances. For single linkage, we have:

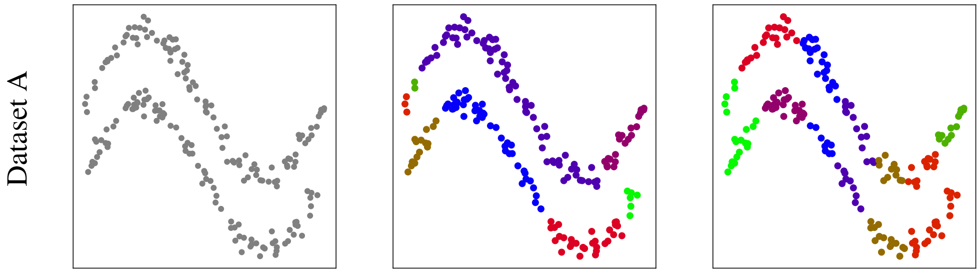
$$D_{\min}(S_1,\ S_2 \cup S_3) = \min \big[\ D_{\min}(S_1, S_2)\ ,\ D_{\min}(S_1, S_3)\ \big],$$

i.e., we can evaluate the new distance in constant time using the previously computed distances. A similar relationship holds for complete linkage. For average linkage, keeping track of the cluster sizes is enough to allow computation in constant time; and for centroid linkage, saving the current centroids and cluster sizes allow it to be computed in $O(n)$ time.

In heirarchical clustering, the linkage choice (cluster dissimilarity measure) implicitly determines what types of clusters will be identified in practice. This indirect dependence means that the resulting clusters can be more flexible, but may be less easily interpreted, than the explicit form of a cluster intrinsic to mixture modeling, for example. Different choices of linkage may be more or less appropriate, depending on what types of clusters we are hoping to find. We illustrate with a few examples.

Example 10-6 : Finding elongated clusters

Consider the data shown below. Visually, we see that these data form two long, connected groups.

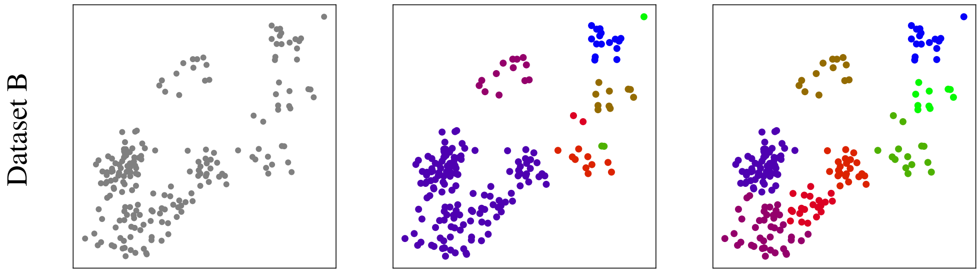


Data Single Linkage Complete Linkage

If we perform agglomerative clustering with single linkage (minimum distance), te procedure will merge two groups if the are close *somewhere*, allowing the agglomerated clusters to respect the differences between the two curves. Complete linkage, in contrast, often merges groups across the two curves.

Example 10-7 : Finding compact clusters

A different data set, with a different desired notion of "clusters", may behave quite differently. Visually, we might think these data are better organized into clumps.



Data Single Linkage Complete Linkage

Then, single linkage will create one large group (purple), and several very small ones (e.g., two clusters with only one data point each). In this particular setting, complete linkage may better match our subjective, visual sense of the data groups.

In general, however, our data are high dimensional, and we cannot visually inspect them to see if the groupings we have identified are appropriate. We need to use our understanding of the domain, and what makes a cohesive set of data in our particular problem, to determine what type of linkage function to apply.

A useful property of linkage clustering is that $D(x, x')$ does not even need to be a proper distance function – it should be symmetric, but need not satisfy other properties such as the triangle inequality. This means it is easy to define $D(\cdot, \cdot)$ on unusual data types, for example, strings or graphs, or to define it even between pairs x, x' for which some feature values are missing. (We shall see an example of this in Section 10.4.)

10.4 Application: Recommendation systems

In many online retail settings, we would like to understand our customers and be able to make product recommendations for them to purchase. This might involve understanding what products are similar and can be offered as alternatives, or understanding genres of books, music or movies to make useful suggestions. We can frame this as a prediction problem – if our customer bought this item, and then rated it, what score would they give it?

For concreteness, consider movie recommendations. Any user will, of course, only rate a given movie once, so any predictive power we obtain must come from generalizing across movies or users. It is also hard to take a standard supervised learning viewpoint – our users might rebel if we require them to tell us a lot of personal information (user features), and it may be impractical to gather item features if our inventory is very large. In such casese we might like to use *only* ratings already input by our users. Intuitively, this matches how we might find a recommendation ourselves – find other people that have similar tastes to ours, and ask them about what they like. This task is sometimes called **collaborative filtering**; it was popularized in part by a competition, with a $1 million prize, sponsored by Netflix in 2005.

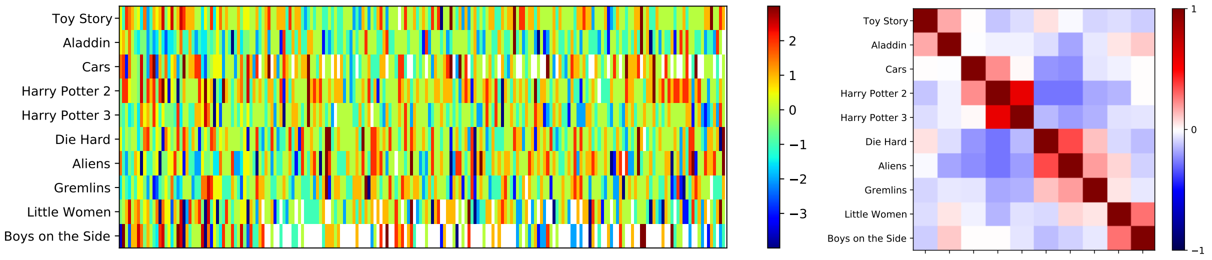
While there are many approaches for performing collaborative filtering (see, e.g., Section 11.3), let us examine how clustering can help with collaborative filtering and rating data. Consider a small movie rating data set (extracted from Harper and Konstan [2015]), consisting of 10 movies and 200 users. Our 10 movies are:

Title	Year	Median	Tags
Toy Story	(1995)	7	Adventure, Animation, Children, Comedy, Fantasy
Aladdin	(1992)	7	Adventure, Animation, Children, Comedy, Musical
Cars	(2006)	6	Animation, Children, Comedy
Harry Potter #2	(2002)	6	Adventure, Fantasy
Harry Potter #3	(2004)	7	Adventure, Fantasy, IMAX
Die Hard	(1988)	7	Action, Crime, Thriller
Aliens	(1986)	7	Action, Adventure, Horror, Sci-Fi
Gremlins	(1984)	6	Comedy, Horror
Little Women	(1994)	6	Drama, Romance
Boys on the Side	(1995)	5	Comedy, Drama

We pre-process the ratings by subtracting the median rating for each movie, and then the median rating for each user. Missing data are a pervasive issue in collaborative filtering, but here, by design, our subset of data is reasonably complete; we discuss missing ratings in more detail later.

The data matrix X^T is shown in Figure 10.11(a). Although it is difficult to see much of a pattern in the data matrix itself, we can see that there is information about similarity in the data. If we compute the correlation coefficients for each pair of movies (shown in Figure 10.11(b)), we can see the ratings are correlated as we might expect. For example, our two Harry Potter films have highly positively correlated ratings (people who liked or disliked one typically felt the same about the other), and similarly the 1980s movies are positively correlated (users who liked “Die Hard” also typically liked “Aliens”, and to a lesser extent, “Gremlins”), but that these two sets are negatively correlated (users who liked one did not like the other as much).

Let us use clustering to understand the structure in these data better. A method commonly used in analyzing gene expression microarray data sets is to perform agglomerative clustering, and use the resulting dendrogram to sort and arrange the array by similarity [Eisen et al., 1998]. Doing so on the ratings data, using complete (“max”) linkage gives a rearrangement of the users shown in Figure 10.12. In this figure, the sub-patterns of groups of users who like and dislike the same movies becomes much



(a) (b)

Figure 10.11: (a) Median-centered ratings for 200 users on 10 movies (white = missing data). (b) Correlation coefficients between the ratings of each pair of movies. Similar movies are often positively correlated by user's ratings (for example, the two "Harry Potter" films), while very dissimilar movies may be negatively correlated (for example, "Harry Potter" and "Die Hard" or "Aliens").

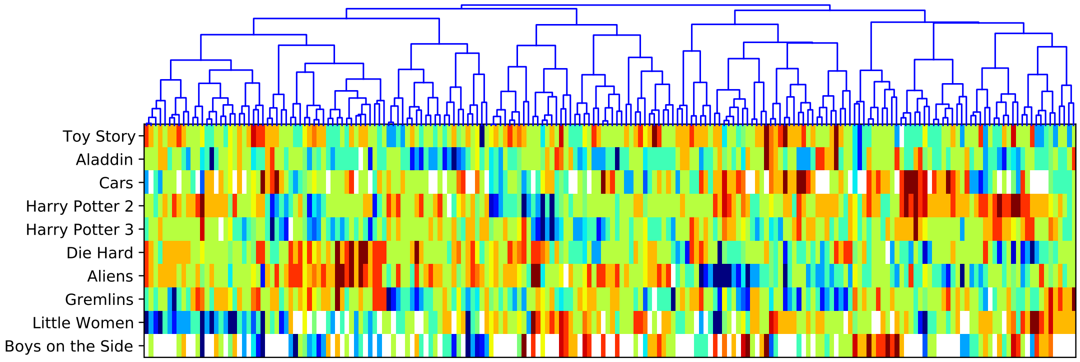


Figure 10.12: Re-ordering the columns of X^T using the heirarchy of groupings found using agglomerative clustering. Compared to Fig. 10.11, similarities in behavior are easier to detect visually, such as the groups of users who did not like "Little Women" or did like "Die Hard" and "Aliens" (left side), or who liked both "Cars" and "Harry Potter" (right side).

clearer. We can see several such groups; at the far left is a group who rated "Little Women" poorly, followed by another group who mostly did not rate that film, but rated "Die Hard" and "Aliens" highly. At the far right, we see a large group who rated both "Cars" and "Harry Potter 2" very highly, etc.

One issue that arises regularly in collaborative filtering problems is the presence of **missing data**, i.e., feature values for a given data point that have not been measured and are unknown. In collaborative filtering, it is quite common for each user to rate only a small subset of the items (movies) in our database. (In the data of Figure 10.11, a few user/item pairs are missing, indicated by white; this subset of users and movies was selected to have few missing values.)

While there are many ways to deal with missing values, in clustering models such as agglomerative clustering which are based on pairwise similarity between the data, we can simply define our measure of similarity in such a way as to sidestep the missing values. For example, a common measure of similarity for collaborative filtering is the **cosine similarity** (expressed here as a dissimilarity D):

$$D(a, b) = 1 - \cos\theta_{ab} = 1 - \frac{a \odot b}{\|a\| \, \|b\|}.$$

When the vectors a, b point in a similar direction from the origin (the angle θ_{ab} between the vectors is small), we have $D(a, b) \approx 0$; if they are exactly opposite, we have $D(a, b) = -1$. The cosine (dis)similarity is easy to extend to vectors a, b with missing values, by (for example) only including the elements of the vectors for which both a and b are non-missing.

Cosine similarity is also closely related to the Pearson correlation score of our measurements, which corresponds to the cosine similarity between $(a - \bar{a})$ and $(b - \bar{b})$, where $\bar{a}$, $\bar{b}$ are the mean values of the vectors a, b.

Missing data.

- Normally lots of missing data
- Using agglomerative clustering, we can measure similarity using a score based only on co-ratings
- Correlation, etc.