CS 5785 – Applied Machine Learning – Lec. 11

Prof. Serge Belongie, Cornell Tech
Scribe: Betty Chou, Jessie Gao, Nish Kumar, Ray Su, Kripa A.

Sept. 2017 (Under construction)

1 Cluster Analysis

1.1 Review from Last Time

Recall that clustering is an unsupervised method. *Central* approaches are prototype-based in that they assume the data is organized into clumps with meaningful centroids. *Pairwise* approaches (also: graph-theoretic) focus on edges connecting the nodes. Central approaches are considered *top-down* since you pick a model before fitting it. Pairwise approaches are considered *bottom-up* as they let structure emerge from the data, making fewer prior assumptions about the model.

1.2 Clustering

Given unlabeled data x_1, x_2,..., x_N assign each x to a a cluster in C.

C(x_i)=2 implies that, "point x_i belongs to cluster 2"

1.3 K-means Clustering

K-means is the most famous clustering algorithm. It is a top-down, iterative, greedy approach that assumes vectorial data and finds a local optimum for K cluster centers. By *top-down*, it means that k is predetermined. By *greedy*, it means that it is an approximation, and computing exact answers is prohibitively expensive. By *local*, it means that it can get "stuck" on local extrema and therefore should always be run many times until stability is reached. By *optimum*, it means that we can write down some cost function. It depends heavily on averaging of vectors. Therefore, it assumes that all of the variables are quantitative, not categorical, as averaging categorical data doesn't make sense.

Assume we have N observations of p-dimensional data denoted x_{ij}; $i = 1, \ldots, N$; and $j = 1, \ldots, p$. K-means computes distances between data points using the Euclidean or L_2 norm:

$$d(x_i, x_{i'}) = \sum_{j=1}^{p}(x_{ij} - x_{i'j})^2 = \|x_i - x_{i'}\|^2$$

K-means assumes that the data is normalized, so the weights on all the features are the same. If feature stretching is desired, it should be done before running K-means.

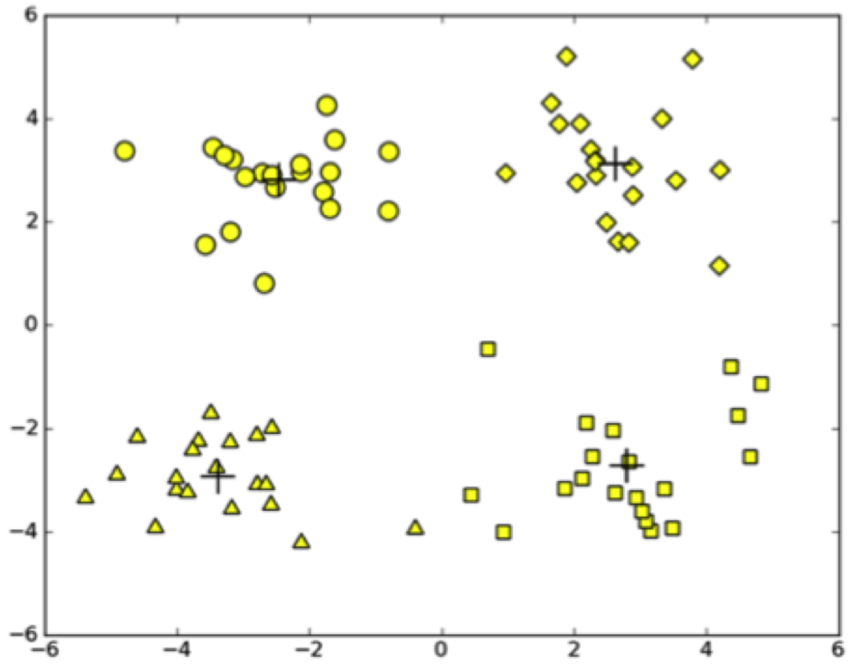


Figure 10.1 **Clusters resulting from k-means clustering. After three iterations, the algorithm converged on these results. Data points with similar shapes are in similar clusters. The cluster centers are marked with a cross.**

Figure 1: Example of K-means clustering assignments.

Suppose we pick $K \ll N$ and choose cluster labels $k \in \{1, ..., K\}$. We assign each observation to exactly one cluster. We define a many-to-one mapping or "encoder" as $k = C(i)$, such that given the index of a data point, the encoder returns the cluster assignment of that point.

Would it be plausible to search over all possible encoders and pick the best one? In practice, that would be prohibitively expensive. The number of possible encoders is given by the Stirling number of the second kind:

$$S(N, K) = \frac{1}{K!} \sum_{k=1}^{K} (-1)^{K-k} \binom{K}{k} k^N$$

For example, if $K = 4$, $S(10, 4) = 34, 105$. That's already quite large. $S(19, 4) > 10^{10}$. Hence the brute force approach is not feasible.

K-means is a greedy, iterative approach. With its *top-down*, *greedy* and *iterative* (and online) approach, K-means examines a small subset of these possible encodings. To see how K-means works, first define the "within-cluster" point scatter as:

$$\begin{aligned} W(C) &= \frac{1}{2} \sum_{k=1}^{K} \sum_{C(i)=k} \sum_{C(i')=k} \|x_i - x_i'\|^2 \\ &= \sum_{k=1}^{K} N_k \sum_{C(i)=k} \|x_i - \bar{x}_k\|^2, \end{aligned}$$

where $\bar{x}_k = (\bar{x}_{ik}, \ldots, \bar{x}_{pk})$ is the mean vector of the kth cluster and $N_k = \sum_{i=1}^{N} I(C(i) = k)$ is the point count of the kth cluster. Given an encoder C, the within-cluster point scatter tells us how spread out the clusters are around their respective centroids. It does this by summing the squared norm of the distance between each pair of points within a cluster, or, equivalently, summing the squared norm of the distance from every point to the mean of the cluster and weighting it by the number of points in that cluster.

The purpose of the K-means algorithm is not classification, K-NN does the classification work. The K-means algorithm (HTF Alg. 14.1) iterates the following 2 steps until the assignments stop changing.

1. Compute the means within each cluster given the current encoding C, call these m_1 ..., m_k.

2. Reassign observations to closest cluster mean.

$$C(i) = \underset{1 \le k \le K}{\arg\min} \|x_i - m_k\|^2$$

The algorithm will start out with some sort of initialization for the means (pick random points from the data as initial means). We iterate until the many-to-one assignment changes stop. Which means the all the data points are already asssigned to closest cluster and there is no change in value for the computed cluster mean in successive iterations. This is illustrated in Fig. 2 with $K = 3$.

We can also look at this animated GIF of the K-means process starting in Fig. 3, 5 and ending up in Fig. 4, 6. (Link to GIF)

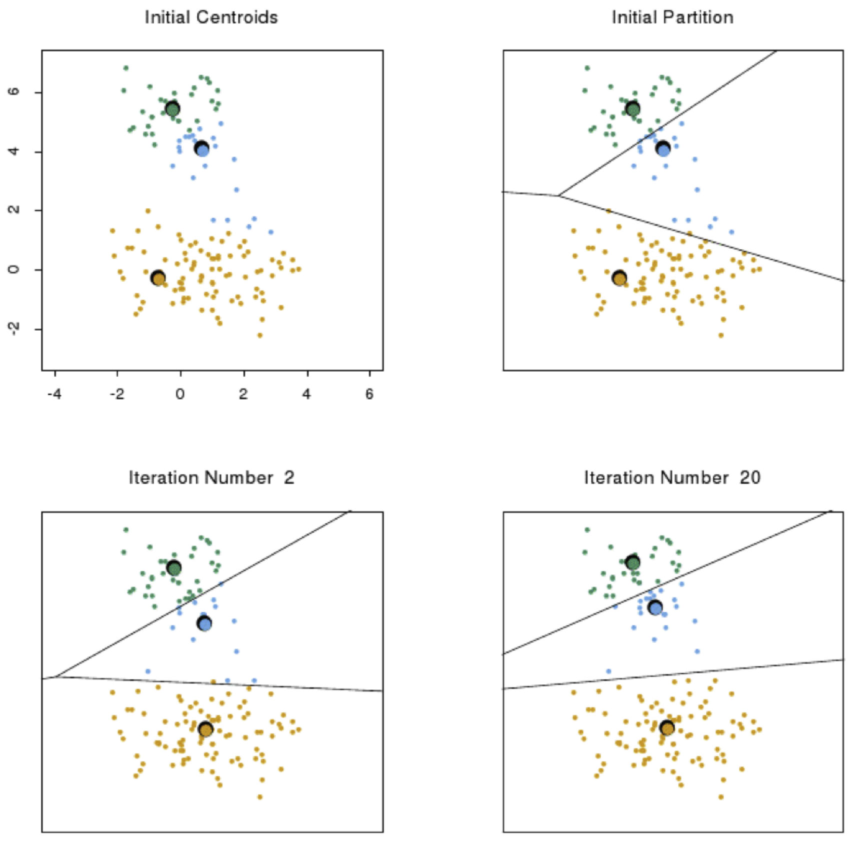


FIGURE 14.6

Figure 2: Illustration of K-means clustering.

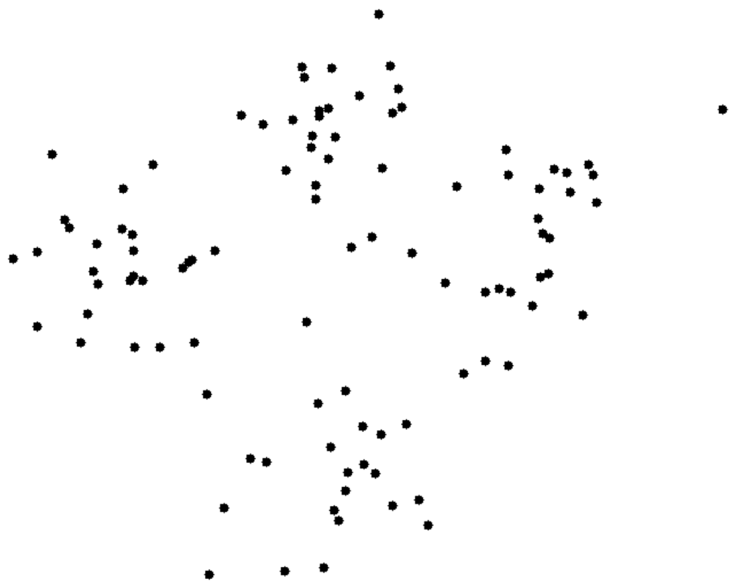


Figure 3: Before K-means clustering (example 1)

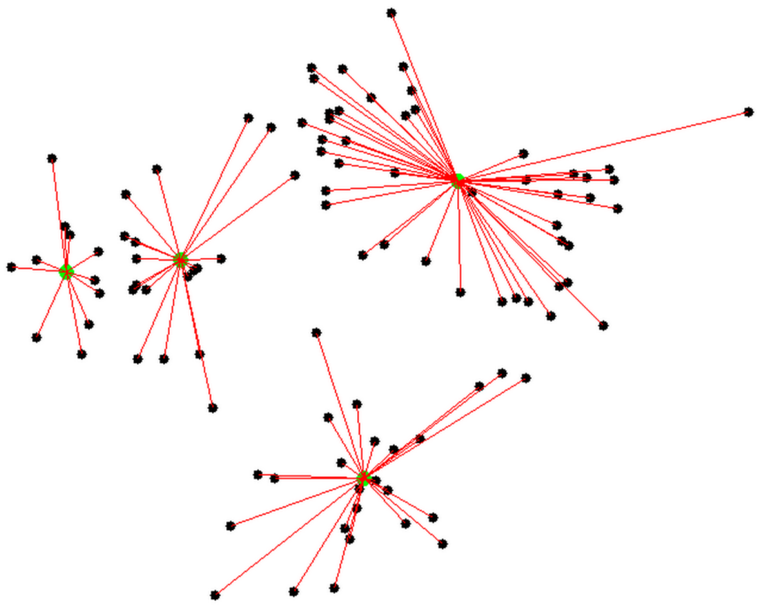


Figure 4: Completed K-means clustering (example 1)

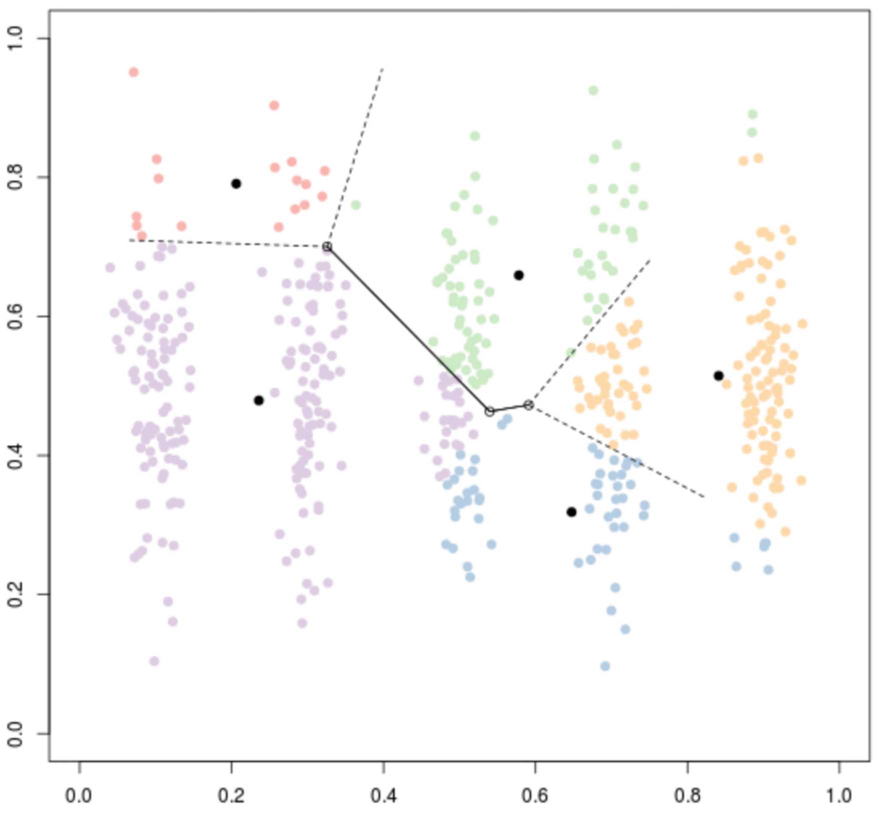


Figure 5: Before K-means clustering (example 2)

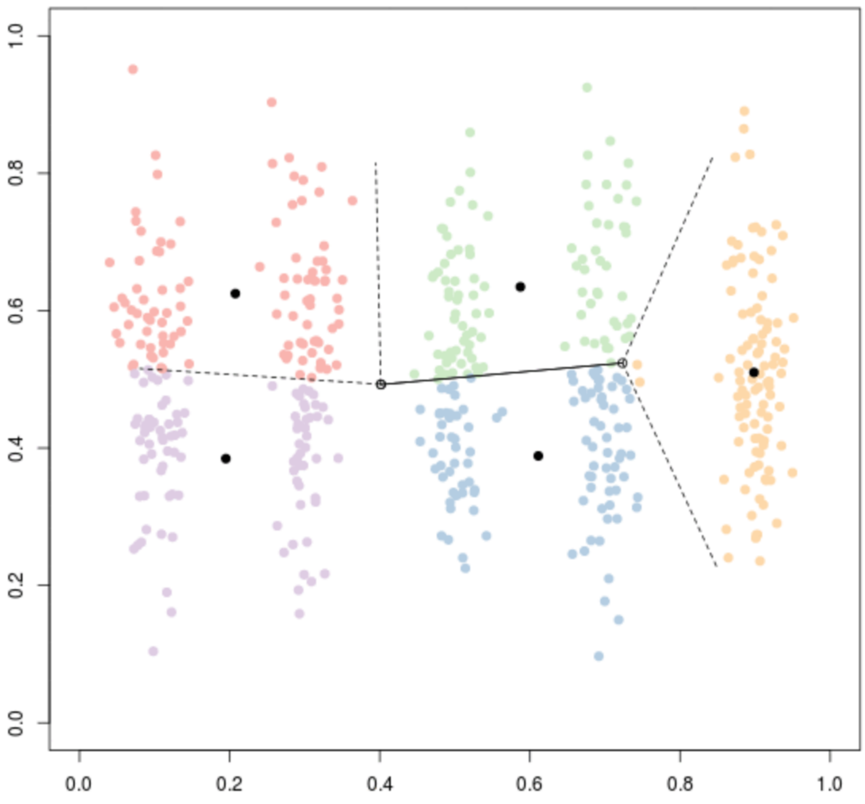


Figure 6: Completed K-means clustering (example 2)

One way to stop is when the labels stop changing. However, this can be tricky if labels keep flip-flopping. One alternative is to stop if the cluster centers or overall cost stop changing according to some threshold. Or, you could just stop after a fixed number of iterations.

When faced with a tie, we can just choose the cluster arbitrarily as we will run the process many times.

Note that K means will always converge in finite time. Also we may not reach global optima as the algorithm may converge to a local optima.

How do we choose K?

There is no optimal K. Since this is an unsupervised problem, we can't use cross validation to choose K (the class label assigned here has no meaning and for each time, it can be of different label). You should definitely try a number of different initializations for each k (maybe 30), as the algorithm may not find the same partitions each time. You could use some heuristic of how often the same partitions (for example, whether two points are assigned in the same cluster) are formed with different initializations. If this wasn't stable you could change K.

This is an instance of a *model order selection problem* in machine learning since adding K will always reduce error. If K becomes N, it can achieve 0 error, but that isn't very useful. Naturally, we need to find a compromise between cost and complexity. There are a variety of information-theoretic or stability-based heuristics one can use for choosing K, but again, there is no single correct value of K, just nice heuristics.

Recall: K-NN is used for classification or regression and is supervised. K-means is for clustering and is unsupervised.

1.4 Fitting Gaussian Mixtures

As we saw above, K-means uses hard cluster assignment on each iteration, so cluster memberships are popping back and forth. Gaussian mixture models (GMM) fitting using the Expectation-Maximization (EM) algorithm offer a soft alternative to K-means. We'll go through the process of GMM fitting using EM for $K = 2$ and $p = 1$ and see how that reduces to K-means. With $K = 2$ we have a bimodal distribution specified as follows:

$$Y_1 \sim \mathcal{N}(\mu_1, \sigma_1^2), \quad Y_2 \sim \mathcal{N}(\mu_2, \sigma_2^2)$$

and

$$Y = (1 - \Delta)Y_1 + \Delta Y_2 \quad \text{with} \quad \Delta \in \{0, 1\},\ Pr(\Delta = 1) = \pi.$$

You flip a coin with probability π that delivers outcome Y_1 or Y_2. This is a *generative representation.* We don't get to observe Y_1 or Y_2 directly since it's unobserved – we can only observe Y, which we model as the outcome of a random experiment. The outcome of the random experiment, denoted here by Δ, is a *latent variable.*

Sometimes, it is better to tie the variances together to avoid over-fitting and lower costs.

For instance, if we were measuring blood sugar for diabetes we would probably see a bimodal distribution with a histogram. But, we don't know what the labels are. We just want to see if there's a cluster structure in the data. The coin-flipping and specified distribution would generate the random data for us.

Let $\phi_\theta(x)$ denote the normal density with parameters $\theta = (\mu, \sigma^2)$. We can write the density of Y as

$$g_Y(y) = (1-\pi)\phi_{\theta_1}(y) + \pi\phi_{\theta_2}(y)$$

We want to fit this model to a dataset of N samples, i.e., to solve for the parameters $\theta = (\pi, \theta_1, \theta_2) = (\pi, \mu_1, \sigma_1^2, \mu_2, \sigma_2^2)$. The log-likelihood is given by

$$\ell(\theta; Z) = \sum_{i=1}^{N} \log[(1-\pi)\phi_{\theta_1}(y_i) + \pi\phi_{\theta_2}(y_i)]$$

where Z denotes the observed data. Direct maximization of this log likelihood is hard due to the sum of terms in the log. Instead, let's pretend we know the value of the latent variable Δ for each point and write:

$$\ell_0(\theta; Z, \Delta) = \sum_{i=1}^{N}[(1-\Delta_i)\log\phi_{\theta_1}(y_i) + \Delta_i \log\phi_{\theta_2}(y_i)] + \sum_{i=1}^{N}[(1-\Delta_i)\log(1-\pi) + \Delta_i \log\pi]$$

Then we can estimate μ_1 and σ_1^2 from the sample mean and variance for the data with $\Delta_i = 0$, similarly for μ_2 and σ_2^2 when $\Delta_i = 1$. We estimate π from the proportion for which $\Delta_i = 1$.

Of course, we don't actually know the Δ_is. We address this problem by making a guess and iterating as follows. For each Δ_i above, we substitute its expected value.

$$\gamma_i(\theta) = E(\Delta_i|\theta, Z) = Pr(\Delta_i = 1|\theta, Z)$$

This is also known as the *responsibility* of the model (model 2, in this case) for observation i. The responsibility for model 1 would be 1 minus this, since there are only two components.

The EM Algorithm (HTF 8.1) for a bimodal GMM is as follows (it looks pretty familiar):

1. Take initial guesses for $\hat{\mu}_1, \hat{\sigma}_2^2, \hat{\mu}_2, \hat{\sigma}_2^2, \hat{\pi}$

2. E-step: compute responsibilities

$$\hat{\gamma_i} = \frac{\hat{\pi}\phi_{\hat{\theta}_2}(y_i)}{(1-\hat{\pi})\phi_{\hat{\theta}_1}(y_i) + \hat{\pi}\phi_{\hat{\theta}_2}(y_i)} \quad \text{for} \quad i = 1, ..., N$$

3. M-step: compute weighted means and variances

$$\hat{\mu}_1 = \frac{\sum_{i=1}^{N}(1-\hat{\gamma_i})y_i}{\sum_{i=1}^{N}(1-\hat{\gamma_i})}$$
$$\hat{\sigma}_1^2 = \frac{\sum_{i=1}^{N}(1-\hat{\gamma_i})(y_i-\hat{\mu}_1)^2}{\sum_{i=1}^{N}(1-\hat{\gamma_i})}$$

   and similarly for $\hat{\mu}_2$, $\hat{\sigma}_2^2$ using $\hat{\gamma}_i$ in place of $(1-\hat{\gamma}_i)$. The mixing probability is given by $\hat{\pi} = \sum_{i=1}^{N}\frac{\hat{\gamma}_i}{N}$.

4. Iterate steps 2 and 3 until convergence.

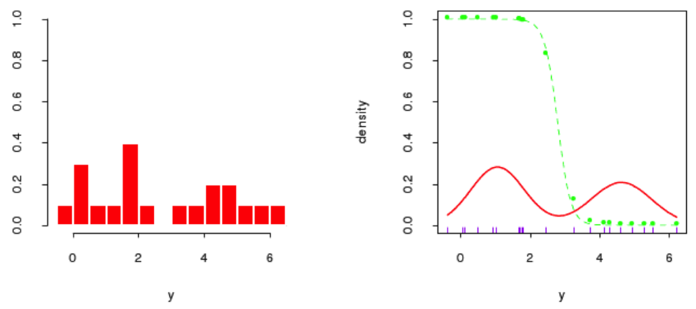


FIGURE 8.5. *Mixture example. (Left panel:) Histogram of data. (Right panel:) Maximum likelihood fit of Gaussian densities (solid red) and responsibility (dotted green) of the left component density for observation y, as a function of y.*

Figure 7: Example of Gaussian Mixture Model fit.

Fig. 7 shows an example. The histogram on the left gives us a rough idea that the data is bimodal. We start with guesses for the means and variances for the two Gaussian components, compute the responsibilities, then compute weighted means and variances conditioned on the current responsibilities and iterate. The green curve on the right represents the responsibilities to one of the modes. In EM, the responsibilities lie in a continuum from 0 to 1. This means that membership is ”soft” compared to K means where membership is either 0 or 1. Only the responsibility for the first mode is shown since the other one is one minus it. For every tick mark along the bottom, the dots on the green curve show the responsibility for the first mode. The averaging takes every data point into account but weights them by the responsibility. Because the responsibility drops off so fast, you probably aren't taking many of the right points into account for the first mean and variance. Fig. 8 plots the log-likelihood as a function of iteration. It appears to be heading into a plateau, which suggests a criterion for stopping.

Fig. 9 shows an example from scikit-learn that, like the above example, is bimodal, but in this case the data is 2D instead of 1D. This requires a slight change to the maximization step to compute the mean vector and covariance matrix instead of the scalar mean and variance. When applying EM to GMM fitting when $p > 1$, we have the option of using a full, diagonal or spherical covariance matrix. Spherical means the covariance matrix has the form $\sigma^2 I$ where I is a $p \times p$ identity matrix. In this example, the covariance matrices are full, and we see that they have adapted to the spread of the data in each mode.

Finally, in Fig. 10 we connect GMM/EM fitting to K-means. If you set all covariance matrices to $\sigma^2 I$ (i.e., spherical) and drive σ^2 to 0, this produces hard assignments. Therefore, when we run K-means,g it's as if we're doing GMM with EM but instead of letting each cluster have its own covariance matrix, they all have to be round covariance matrices and we make σ^2 tiny so there's no longer any notion of shared membership between clusters.

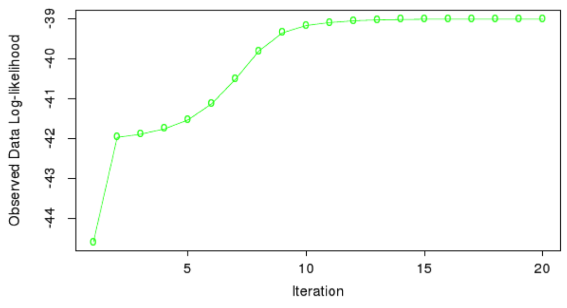
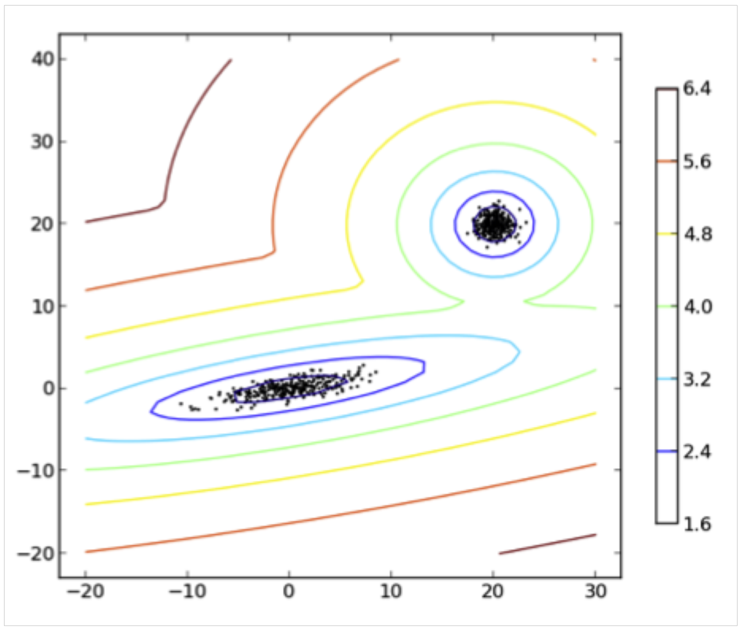


FIGURE 8.6. *EM algorithm: observed data log-likelihood as a function of the iteration number.*

Figure 8: Log-likelihood vs. iterations of EM.



[scikit-learn user guide,
Release 0.11]

Figure 9: Scikit learn example.

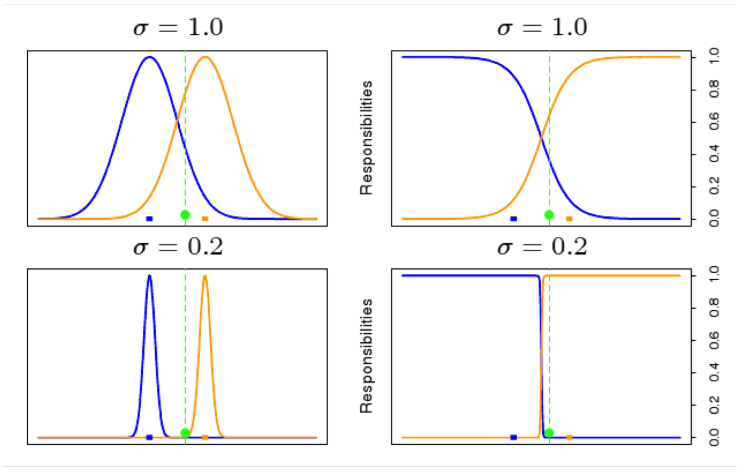


FIGURE 14.7

Figure 10: Left: Gaussian densities for two modes with the same means but different variances. Right: plot of responsibilities. In the case where $\sigma = 1.0$ the green dot picks off shared responsibilities between the two modes. When $\sigma = 0.2$ the green dot gets a responsiblity of nearly 1 for the right mode and nearly 0 for the other since it lies just a hair to the right of the midpoint between the two means.

As a side note, K-means is sometimes used to initialize GMM/EM fitting.