15 Clustering

Clustering is an **unsupervised** learning problem in which our goal is to discover "clusters" in the data. A cluster is a collection of data that are similar in some way.

Clustering is often used for several different problems. For example, a market researcher might want to identify distinct groups of the population with similar preferences and desires. When working with documents you might want to find clusters of documents based on the occurrence frequency of certain words. For example, this might allow one to discover financial documents, legal documents, or email from friends. Working with image collections you might find clusters of images which are images of people versus images of buildings. Often when we are given large amounts of complicated data we want to look for some underlying structure in the data, which might reflect certain *natural kinds* within the training data. Clustering can also be used to compress data, by replacing all of the elements in a cluster with a single representative element.

15.1 K-means Clustering

We begin with a simple method called K-means. Given N input data vectors $\{\mathbf{y}_i\}_{i=1}^N$, we wish to label each vector as belonging to one of K clusters. This labeling will be done via a binary matrix $\mathbf{L}$, the elements of which are given by

$$L_{i,j} = \begin{cases} 1 & \text{if data point } i \text{ belongs to cluster } j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

The clustering is mutually exclusive. Each data vector i can only be assigned to only cluster: $\sum_{j=1}^K L_{i,j} = 1$. Along the way, we will also be estimating a center $\mathbf{c}_j$ for each cluster.

The full objective function for K-means clustering is:

$$E(\mathbf{c}, \mathbf{L}) = \sum_{i,j} L_{i,j} ||\mathbf{y}_i - \mathbf{c}_j||^2 \tag{2}$$

This objective function penalizes the distance between each data point and the center of the cluster to which it is assigned. Hence, to minimize this error, we want to bring the cluster centers close to the data it has been assigned, and we also want to assign the data to nearby centers.

This objective function cannot be optimized in closed-form, and so an iterative method is required. It includes discrete variables (the labels $\mathbf{L}$), and so gradient-based methods aren't directly applicable. Instead, we use a strategy called **coordinate descent**, in which we alternate between closed-form optimization of one set of variables holding the other variables fixed. That is, we first pick initial values, then we alternate between updating the labels for the current centers, and then updating the centers for the current labels.

Here is the K-means algorithm:

pick initial values for $\mathbf{L}$ and $\mathbf{c}_{1:K}$
loop
  // Labeling update: set $\mathbf{L} \leftarrow \arg\min_{\mathbf{L}} E(\mathbf{c}, \mathbf{L})$
  for each data point i **do**
    $j \leftarrow \arg\min_j ||\mathbf{y}_i - \mathbf{c}_j||^2$
    $L_{i,j} = 1$
    $L_{i,a} = 0$ for all $a \neq j$
  end for
  // Centers update: set $\mathbf{c} \leftarrow \arg\min_{\mathbf{c}} E(\mathbf{c}, \mathbf{L})$
  for each center j **do**
    $\mathbf{c}_j \leftarrow \frac{\sum_i L_{i,j} \mathbf{y}_i}{\sum_i L_{i,j}}$
  end for end loop

Each step of the optimization is guaranteed to lower the objective function until the algorithm converges (you should be able to show that each step is optimal.) However, there is no guarantee that the algorithm will find the global optimum and indeed it may easily get trapped in a poor local minima.

Initialization. The algorithm is sensitive to initialization, and poor initialization can sometimes lead to very poor results. Here are a few strategies that can be used to initialize the algorithm:

1. **Random labeling:** Initialize the labeling $\mathbf{L}$ randomly, and then run the center-update step to determine the initial centers. This approach is not recommended because the initial centers will likely end up just being very close to the mean of the data.

2. **Random initial centers:** We could try to place initial center locations randomly, e.g., by random sampling in the bounding box of the data. However, it is very likely that some of the centers will fall into empty regions of the feature space, and will therefore be assigned no data. Getting a good initialization this way can be difficult.

3. **Random data points as centers:** This method works much better: use a random subset of the data as the initial center locations.

4. **K-medoids clustering:** This will be described below.

5. **Multiple restarts.** In multiple restarts, we run K-means multiple times, each time with a different random initialization (using one of the above methods). We then take the best clustering out of all of the runs, based on the value of the objective function above in Equation (2).

6. **K-Means++:** k-means++ chooses the initial centers to be relatively far from one another. That is, (1) Choose a center at random. (2) Then compute the distance between each data point $D(x)$ and the nearest existing center. (3) Then choose the next centre from among the data points according to probability proportional to $D(x)^2$. And once the new center is chosen, repeat (2)-(3) until k centers have been chosen. Then optimize k-means as above.

Another key question is how one chooses the number of clusters, i.e., K. A common approach is to fix K based on some prior knowledge or computational constraints. One can also try different values of K, adding another term to to the objective function to penalize model complexity.

15.2 K-medoids Clustering

(The material in this section is not required for this course.)

K-medoids clustering is a variant of K-means with the additional constraint that the cluster centers must be drawn from the data. The following algorithm, called Farthest First Traversal, or Hochbaum-Shmoys, is simple and effective:

Randomly select a data point $\mathbf{y}_i$ as the first cluster center: $\mathbf{c}_1 \leftarrow \mathbf{y}_i$
for $j = 2$ to K
  Find the data point furthest from all existing centers:
  $i \leftarrow \arg\max_i \min_{k<j} ||\mathbf{y}_i - \mathbf{c}_k||^2$
  $\mathbf{c}_j \leftarrow \mathbf{y}_i$
end for
Label all remaining data points according to their nearest centers (as in k-means)

This algorithm provides a quality guarantee: it gives a clustering that is no worse than twice the error of the optimal clustering.

K-medoids clustering can also be improved by coordinate descent. The labeling step is the same as in K-means. However, the cluster updates must be done by brute-force search for each candidate cluster center update.

15.3 Mixtures of Gaussians

The Mixtures-of-Gaussians (MoG) model is a generalization of K-means clustering. Whereas K-means clustering works for clusters that are more or less spherical, the MoG model can handle oblong clusters and overlapping clusters. The K-means algorithm does an excellent job when clusters are well separated, but not when the clusters overlap. MoG algorithms compute a "soft," probabilistic clustering which allows the algorithm to better handle overlapping clusters. Finally, the MoG model is probabilistic, and so it can be used to learn probability distributions from data.

The MoG model consists of K Gaussian distributions, each with their own means and covariances $\{(\mu_j, \mathbf{K}_j)\}$. Each Gaussian also has an associated (prior) probability a_j, such that $\sum_j a_j = 1$. That is, the probabilities a_j will represent the fraction of the data that are assigned to (or generated

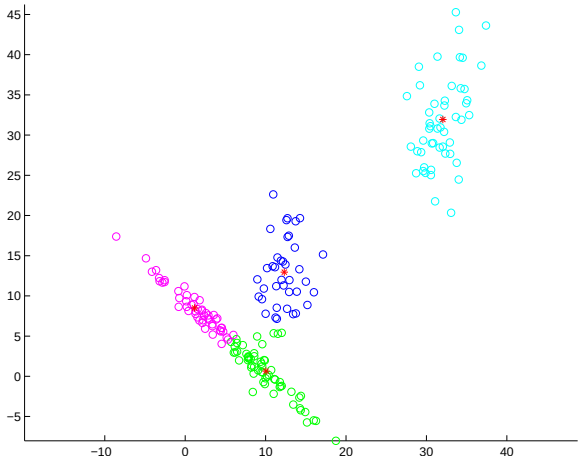

Figure 1: K-means applied to a dataset sampled from three Gaussian distributions. Each data assigned to each cluster are drawn as circles with distinct colours. The cluster centers are shown as red stars.

by) the different Gaussian components. As a shorthand, we will write all the model parameters with a single variable, i.e., $\theta = \{a_{1:K}, \mu_{1:K}, \mathbf{K}_{1:K}\}$. When used for clustering, the idea is that each Gaussian component in the mixture should correspond to a single cluster.

The complete probabilistic model comprises the prior probabilities of each Gaussian component, and Gaussian likelihood over the data (or feature) space for each component:

$$P(L = j|\theta) = a_j \tag{3}$$

$$p(\mathbf{y}|\theta, L = j) = G(\mathbf{y};\, \mu_j, \mathbf{K}_j) \tag{4}$$

To sample a single data point from this (generative) model, we first randomly select a Gaussian component according to their prior probabilities $\{a_j\}$, and then we randomly sample from the corresponding Gaussian component. The likelihood of a single data point can be derived by the product rule and the sum rule as follows:

$$p(\mathbf{y}|\theta) = \sum_{j=1}^{K} p(\mathbf{y}, L = j|\theta) \tag{5}$$

$$= \sum_{j=1}^{K} p(\mathbf{y}|L = j, \theta)\, P(L = j|\theta) \tag{6}$$

$$= \sum_{j=1}^{K} a_j \frac{1}{\sqrt{(2\pi)^D |K_j|}} e^{-\frac{1}{2}(\mathbf{y}-\mu_j)^T \mathbf{K}_j^{-1}(\mathbf{y}-\mu_j)} \tag{7}$$

where D is the dimension of data vectors. This model can be interpreted as a linear combination (or blend) of Gaussians: we get a multimodal distribution by adding together unimodal Gaussians.

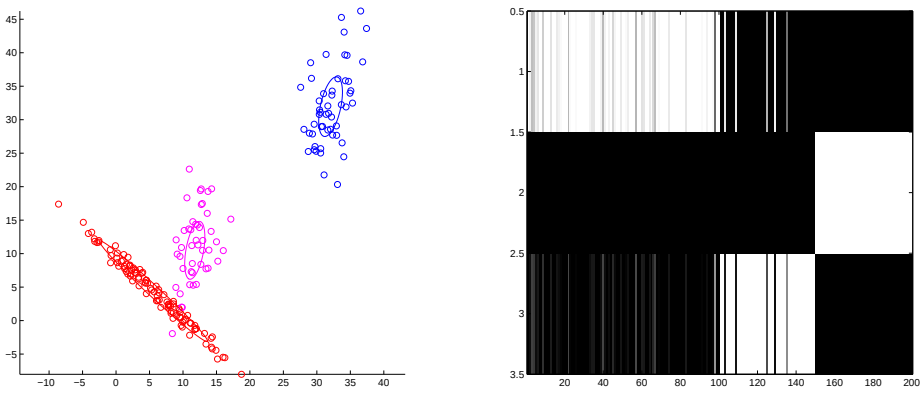

Figure 2: Mixture of Gaussians model applied to a dataset generated from three Gaussians. The resulting γ is visualized on the right. The data points are shown as colored circles. The color is determined by the cluster with the highest posterior assignment probability γ_{ij}. One standard deviation ellipses are shown for each Gaussian. Note that the blue points are well isolated and there is little ambiguity in their assignments. The other two distributions overlap, and one can see how the orientation and eccentricity of the covariance structure (the ellipses) influence the assignment probabilities.

Interestingly, the MoG model is similar to the Gaussian Class-Conditional model that we used for classification; the difference is that the class labels will no longer be included in the training set.

In general, the approach of building models by mixtures is quite general and can be used for many other types of distributions as well, for example, we could build a mixture of Student-t distributions, or a mixture of a Gaussian and a uniform, and so on.

15.3.1 Learning

Given a data set $\mathbf{y}_{1:N}$, where each data point is assumed to be drawn independently from the model, we learn the model parameters, θ, by minimizing the negative log-likelihood of the data:

$$\begin{aligned} \mathcal{L}(\theta) &= -\ln p(\mathbf{y}_{1:N}|\theta) && (8) \\ &= -\sum_i \ln p(\mathbf{y}_i|\theta) && (9) \end{aligned}$$

Note that this is a constrained optimization, since we require $a_j \geq 0$ and $\sum_j a_j = 1$. Furthermore, $\mathbf{K}_j$ must be symmetric, positive-definite matrix to be a covariance matrix. Unfortunately, this optimization cannot be performed in closed-form.

One approach is to use gradient descent to optimization by gradient descent. There are a few issues associated with doing so. First, some care is required to avoid numerical issues, as discussed below. Second, this learning is a constrained optimization, due to constraints on the values of the

a's. One solution is to project onto the constraints during optimization: at each gradient descent step (and inside the line search loop), we clamp all negative a values to zero and renormalize the a's so that they sum to one. Another option is to reparameterize the problem to be unconstrained. Specifically, we define new variables β_j, and define the a's as functions of the βs, e.g.,

$$a_j(\beta) = \frac{e^{\beta_j}}{\sum_{j=1}^{K} e^{\beta_j}} \tag{10}$$

This definition ensures that, for any choice of the βs, the as will satisfy the constraints. We substitute this expression into the model definition and then optimize for the βs instead of the as with gradient descent. Similarly, we can enforce the constraints on the covariance matrix by reparameterization; this is normally done using a upper-triangular matrix $\mathbf{U}$ such that $\mathbf{K} = \mathbf{U}^T\mathbf{U}$.

An alternative to gradient descent is the **Expectation-Maximization** algorithm, or EM. EM is a quite general algorithm for "hidden variable" problems; in this case, the labels L are "hidden" (or "unobserved"). In EM, we define a probabilistic labeling variable $\gamma_{i,j}$. The variable $\gamma_{i,j}$ corresponds to the probability that data point i came from cluster j: $\gamma_{i,j}$ is meant to estimate $P(L = j|\mathbf{y}_i)$. In EM, we optimize both θ and γ together. The algorithm alternates between the "E-step" which updates the γs, and the "M-step" which updates the model parameters θ.

pick initial values for γ and θ
loop
  E-step:
  for each data point i **do**
    $\gamma_{i,j} \leftarrow P(L = j|\mathbf{y}_i, \theta)$
  end for
  M-step:
  for each cluster j **do**
    $a_j \leftarrow \frac{\sum_i \gamma_{i,j}}{N}$
    $\mu_j \leftarrow \frac{\sum_i \gamma_{i,j}\mathbf{y}_i}{\sum_i \gamma_{i,j}}$
    $\mathbf{K}_j \leftarrow \frac{\sum_i \gamma_{i,j}(\mathbf{y}_i-\mu_j)(\mathbf{y}_i-\mu_j)^T}{\sum_i \gamma_{i,j}}$
  end for
end loop

Note that the E-step is the same as classification in the Gaussian Class-Conditional model.

The EM algorithm is a local optimization algorithm, and so the results will depend on initialization. Initialization strategies similar to those used for K-means above can be used.

15.3.2 Numerical issues

Exponentiating very small negative numbers can often lead to underflow when implemented in floating-point arithmetic, e.g., e^{-A} will give zero for large A, and $\ln e^{-A}$ will give an error (or `-Inf`) whereas it should return $-A$. These issues will often cause machine learning algorithms to fail; MoG has several steps which are susceptible. Fortunately, there are some simple tricks that can be used.

1. Many computations can be performed directly in the log domain. For example, it may be more stable to compute
$$ae^b \tag{11}$$
as
$$e^{\ln a + b} \tag{12}$$
This avoids issues where b is so small that e^b evaluates to zero in floating point, but ae^b is much greater than zero.

2. When computing an expression of the form:
$$\frac{e^{-\beta_j}}{\sum_j e^{-\beta_j}} \tag{13}$$
large values of β could lead to the above expression being zero for all j, even though the expression must sum to one. This may arise, for example, when computing the γ updates, which have the above form. The solution is to make use of the identity:
$$\frac{e^{-\beta_j}}{\sum_j e^{-\beta_j}} = \frac{e^{-\beta_j + C}}{\sum_j e^{-\beta_j + C}} \tag{14}$$
for any value of C. We can choose C to prevent underflow; a suitable choice is $C = \min_j \beta_j$.

3. Underflow can also occur when evaluating
$$\ln \sum_i e^{-\beta_j} \tag{15}$$
which can be fixed by using the identity
$$\ln \sum_i e^{-\beta_j} = \left(\ln \sum_i e^{-\beta_j + C} \right) - C \tag{16}$$

15.3.3 The Free Energy

Amazingly, EM optimizes the log-likelihood, which doesn't even have a γ parameter. In order to understand the EM algorithm and why it works, it is helpful to introduce a quantity called the **Free Energy**:

$$\begin{aligned}
\mathcal{F}(\theta, \gamma) &= -\sum_{i,j} \gamma_{i,j} \ln p(\mathbf{y}_i, L = j|\theta) + \sum_{i,j} \gamma_{i,j} \ln \gamma_{i,j} && (17)\\
&= \frac{1}{2} \sum_{i,j} \gamma_{i,j} (\mathbf{y}_i - \mu_j)^T \mathbf{K}_j^{-1} (\mathbf{y}_i - \mu_j) && (18)\\
&\quad + \frac{1}{2} \sum_{i,j} \gamma_{i,j} \ln (2\pi)^D |\mathbf{K}_j| - \sum_{i,j} \gamma_{i,j} \ln a_j && (19)\\
&\quad + \sum_{i,j} \gamma_{i,j} \ln \gamma_{i,j} && (20)
\end{aligned}$$

The EM algorithm is a coordinate descent algorithm for optimizing the free energy, subject to the constraint that $\sum_j \gamma_{i,j} = 1$ and the constraints on a. In other words, EM can be written compactly as:

pick initial values for γ and θ
loop
E-step:
$\gamma \leftarrow \arg\min_\gamma \mathcal{F}(\theta, \gamma)$
M-step:
$\theta \leftarrow \arg\min_\theta \mathcal{F}(\theta, \gamma)$
end loop

However, the free energy is different from the negative log-likelihood $\mathcal{L}(\theta)$ that we initially set out to minimize. Fortunately, the free energy has the following important properties:

- When the value of γ is optimal, the Free Energy is equal to the negative log-likelihood:

$$\mathcal{L}(\theta) = \min_\gamma \mathcal{F}(\theta, \gamma) \quad (21)$$

  We can use this fact to evaluate the negative log-likelihood simply by running an E-step and then computing the free energy. In fact, this is often more efficient than directly computing the negative log-likelihood. The proof is given in the next section.

- The minima of the free energy are also minima of the negative log-likelihood:

$$\min_\theta \mathcal{L}(\theta) = \min_{\theta,\gamma} \mathcal{F}(\theta, \gamma) \quad (22)$$

  This follows from the previous property. Hence, **optimizing the free energy is the same as optimizing the negative log-likelihood.**

- The Free Energy is an upper-bound on the negative log-likelihood:

$$\mathcal{F}(\theta, \gamma) \geq \mathcal{L}(\theta) \tag{23}$$

  for all values of γ. This observation gives a sanity check for debugging the free energy computation.

The Free Energy also provides a very helpful tool for debugging: any step of an implementation that increases the free energy must be incorrect. The term Free Energy arises from its original definition in statistical physics.

15.3.4 Proofs

This content of this section is not required material for this course and you may skip it. Here we outline proofs for the key features of the free energy.

EM updates. The steps of the EM algorithm may be derived by solving $\arg\min_\gamma \mathcal{F}(\theta, \gamma)$ and $\arg\min_\theta \mathcal{F}(\theta, \gamma)$. In most cases, the derivations generalize familiar ones, e.g., weighted least-squares. The a and γ parameters are multinomial distributions, and optimization of them requires Lagrange multipliers or reparameterization. One may ignore the positivity constraint, as it turns out to be automatically satisfied. The details will be skipped here.

Equality after the E-step. The E-step computes the optimal value for γ:

$$\gamma^* \leftarrow \arg\min_\gamma \mathcal{F}(\theta, \gamma) \tag{24}$$

which is given by:

$$\gamma^*_{i,j} = P(L = j | \mathbf{y}_i) \tag{25}$$

Substituting this into the Free Energy gives:

$$\begin{aligned}
\mathcal{F}(\theta, \gamma^*) &= -\sum_{i,j} P(L = j|\mathbf{y}_i) \ln \frac{p(\mathbf{y}_i, L = j)}{P(L = j|\mathbf{y}_i)} && (26)\\
&= -\sum_{i,j} P(L = j|\mathbf{y}_i) \ln p(\mathbf{y}_i) && (27)\\
&= -\sum_i \left(\ln p(\mathbf{y}_i) \sum_j P(L = j|\mathbf{y}_i) \right) && (28)\\
&= -\sum_i \ln p(\mathbf{y}_i) && (29)\\
&= \mathcal{L}(\theta) && (30)
\end{aligned}$$

Hence,

$$\mathcal{L}(\theta) = \min_\gamma \mathcal{F}(\theta, \gamma) \tag{31}$$

Bound. An important building block in proving that $\mathcal{F}(\theta, \gamma) \geq \mathcal{L}(\theta)$ is **Jensen's Inequality**, which applies since $\ln$ is a "concave" function and $\sum_j b_j = 1, b_j \geq 0$.

$$\ln \sum_j b_j x_j \geq \sum_j b_j \ln x_j, \quad \text{or} \tag{32}$$

$$-\ln \sum_j b_j x_j \leq -\sum_j b_j \ln x_j \tag{33}$$

We will not prove this here.

We can then derive the bound as follows, dropping the dependence on θ for brevity:

$$\mathcal{L}(\theta) = -\sum_i \ln \sum_j p(\mathbf{y}_i, L = j) \tag{34}$$

$$= -\sum_i \ln \sum_j \frac{\gamma_{i,j}}{\gamma_{i,j}} p(\mathbf{y}_i, L = j) \tag{35}$$

$$\leq -\sum_{i,j} \gamma_{i,j} \ln \frac{p(\mathbf{y}_i, L = j)}{\gamma_{i,j}} \tag{36}$$

$$= \mathcal{F}(\theta, \gamma) \tag{37}$$

15.3.5 Relation to K-means

It should be clear that the K-means algorithm is very closely related to EM. In fact, EM reduces to K-means if we make the following restrictions on the model:

- The class probabilities are equal: $a_j = \frac{1}{K}$.
- The Gaussians are spherical with identical variances: $\mathbf{K}_j = \sigma^2 \mathbf{I}$ for all j.
- The Gaussian variances are infinitesimal, i.e., we consider the algorithm in the limit as $\sigma^2 \to 0$. This causes the optimal values for γ to be binary, since, if j is the nearest class, $\lim_{\sigma^2 \to 0} P(L = j | \mathbf{y}_i) = 1$.

With these modifications, the Free Energy becomes equivalent to the K-means objective function, up to constant values, and the EM algorithm becomes identical to K-means.

15.3.6 Degeneracy

There is a degeneracy in the MoG objective function. Suppose we center one Gaussian at one of the data points, so that $\mathbf{c}_j = \mathbf{y}_i$. The error for this data point will be zero, and by reducing the variance of this Gaussian, we can always increase the likelihood of the data. In the limit as this Gaussian's variance goes to zero, the data likelihood goes to infinity. Hence, some effort may be required to avoid this situation. This degeneracy can also be avoided by using a more Bayesian form of the algorithm, e.g., marginalizing out the cluster centers rather than estimating them.

15.4 Determining the number of clusters

Determining the value of K is a model selection problem: we want to determine the most-likely value of K given the data. Cross validation is not appropriate here, since we do not have any supervision (e.g., correct labels from a subset of the data). Bayesian model selection can be employed, e.g., by maximizing

$$K^* = \arg\max_K P(K|\mathbf{y}_{1:N}) = \arg\max_K \int p(K, \theta|\mathbf{y}_{1:N}) d\theta \tag{38}$$

where θ are the model parameters. This evaluation is somewhat mathematically-involved. A very coarse approximation to this computation is Bayesian Information Criterion (BIC).

