Week 8: Expectation Maximization

Instructor: Sergey Levine

1 Probabilistic Clustering

So far, we discussed clustering algorithms that involve a hard assignment of each datapoint to a cluster, typically based on its proximity to other points in that cluster. However, simply assigning points to the nearest cluster is not always adequate to capture more complex structure. For example, the lecture slides show an example where one cluster is much larger and less dense than another. We'll discuss a probabilistic clustering method that can capture this by introducing two new ingredients: soft assignment of points $\mathbf{x}_i$ to each cluster, and using additional parameters to discuss the shape of a cluster, so as to capture the fact that clusters can be large, small, sparse, or dense.

2 A Generative Mixture Model

In order to design a probabilistic clustering model, we have to think about a probabilistic process that gives rise to the data. In the case of clustering, the consists of vectors $\mathbf{x}_i \in \mathbb{R}^D$, but we believe that this data originates from a set of discrete clusters. So a natural choice for the probabilistic process looks like this: if you want to generate a new datapoint $\mathbf{x}$, choose a cluster index y at random from the cluster prior $p(y)$, and then generate the point $\mathbf{x}$ from the distribution associated with that cluster, $p(\mathbf{x}|y)$.

You might have noticed at this point that the model looks a lot like naïve Bayes. Indeed, since we have $y \in \{1, \dots, K\}$, and $\mathbf{x} \in \mathbb{R}^D$, a reasonable choice for $p(y)$ would be a multinomial distribution, and a reasonable choice for $p(\mathbf{x}|y)$ might be a product of independent Gaussians, each given by

$$p(\mathbf{x}_j|y=k) = \frac{1}{\sigma_{k,j}\sqrt{2\pi}} \exp\left(-\frac{(\mathbf{x}_j - \mu_{k,j})^2}{2\sigma_{k,j}^2}\right).$$

This model allows us to describe clusters that are different shape, since we can assign a different variance $\sigma_{k,j}^2$ to each dimension j of each cluster k. So, for example, if $\mathbf{x}$ has two dimensions, and we want to create a cluster that is tall and skinny, we might set $\sigma_{k,1}^2 = 0.1$ and $\sigma_{k,2}^2 = 10.0$. Note that so far we are simply describing a generative model: a model that can generate datapoints that originate from a mixture of clusters. We have not yet discussed how this model can be trained on data.

Before we do, let's introduce a slightly more sophisticated way to handle the condition distribution $p(\mathbf{x}|y)$. In naïve Bayes, we assume that all dimensions of $\mathbf{x}$ are independent. This corresponds to ellipsoidal clusters, where the major and minor axes of the ellipsoids are aligned with the axes. But what if we also want to model clusters that are not axis-aligned? Such clusters have *covariance* between different dimensions of $\mathbf{x}$: that is, if we have a cluster in 2D that oriented diagonally, what that really means is that $\mathbf{x}_1$ and $\mathbf{x}_2$ are strongly correlated: for example, if the major axis is oriented along the line $\mathbf{x}_1 = \mathbf{x}_2$, what that means is that, as $\mathbf{x}_1$ gets bigger, $\mathbf{x}_2$ is likely to also get bigger. That means that $\mathbf{x}_1$ and $\mathbf{x}_2$ have positive covariance. In order to model covariance between the dimensions of $\mathbf{x}$, we need to use a multivariate Gaussian distribution, given by

$$p(\mathbf{x}|y=k)\frac{1}{\sqrt{(2\pi)^D|\Sigma_k|}}\exp\left(-\frac{1}{2}(\mathbf{x}-\mu_k)^T\Sigma_k^{-1}(\mathbf{x}-\mu_k)\right).$$

Note that the form of this distribution is very similar to the univariate Gaussian, except that the exponent now includes a matrix Σ_k, instead of a scalar variance σ_k^2. The entries in the matrix Σ_k are the covariances of the various dimensions of $\mathbf{x}$. Entries along the diagonal are the variances of each dimension, so

$$\Sigma_{k,i,i} = E[(\mathbf{x}_i - \mu_{k,i})^2].$$

Off-diagonal entries are covariances between different dimensions:

$$\Sigma_{k,i,j} = E[(\mathbf{x}_i - \mu_{k,i}) \cdot (\mathbf{x}_j - \mu_{k,j})].$$

Just like with the univariate Gaussian, we can estimate the parameters of a multivariate Gaussian using MLE by differentiating the probability density function and setting the derive to zero. The corresponding solution for the mean is given by

$$\mu = \frac{1}{N}\sum_{i=1}^{N}\mathbf{x}_i,$$

and the solution for the covariance is

$$\Sigma = \frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i - \mu)(\mathbf{x}_i - \mu)^T.$$

If we want to fit a conditional model of the form $p(\mathbf{x}|y)$, where there is a different multivariate Gaussian for each value of y, we can do that also. If we have labels y_i for each datapoint $\mathbf{x}_i$, we can do this analogously to the method we used for naïve Bayes:

$$\mu_k = \frac{1}{\text{Count}(y_i = k)}\sum_{i:y_i=k}\mathbf{x}_i$$

$$\Sigma_k = \frac{1}{\text{Count}(y_i = k)}\sum_{i:y_i=k}(\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^T$$

But how can we do this in the case of clustering, when the labels y_i are unknown?

3 Expectation Maximization

To start thinking about this, let's imagine a simplified scenario: imagine that we already have a model $p(\mathbf{x}, y)$ defined by K multivariate Gaussians and a prior $p(y)$, and now we are given a dataset $\mathbf{x}_1, \ldots, \mathbf{x}_N$ without labels. How can we recover a guess about the labels? Well, this is simply a prediction problem, and our (already trained) model can give us a distribution over the label of each datapoint. Just like we did with naïve Bayes, we can apply Bayes rule to get:

$$p(y|\mathbf{x}) = \frac{p(\mathbf{x}, y)}{p(\mathbf{x})} = \frac{p(\mathbf{x}|y)p(y)}{p(\mathbf{x})} \propto p(\mathbf{x}|y)p(y).$$

So if we want to know $p(y_i = k|\mathbf{x}_i)$, we just compute $p(\mathbf{x}_i|y_i = k)p(y_i = k)$ for each value of $k \in \{1, \ldots, K\}$, and then normalize so that the probabilities of the K different labels sum to one. As we learned before, this is called inference. So we first evaluate the unnormalized probability for each datapoint:

$$\tilde{w}_{ik} = p(y_i = k) \frac{1}{\sqrt{(2\pi)^D |\Sigma_k|}} \exp\left(-\frac{1}{2}(\mathbf{x}_i - \mu_k)^T \Sigma_k^{-1} (\mathbf{x}_i - \mu_k)\right),$$

and then normalize to get

$$w_{ik} = \frac{\tilde{w}_{ik}}{\sum_{k'=1}^{K} \tilde{w}_{ik'}}.$$

So if we are lucky enough to already have a trained model, we can invert that model using Bayes rule to obtain a distribution over labels for each datapoint.

Now, let's assume the opposite: we have probabilistic assignments of each datapoint to clusters, in the form of weights $w_{ik} = p(y_i = k|\mathbf{x}_i)$, and we wish to fit the model parameters: the prior $p(y)$ and the distributions $p(\mathbf{x}|y = k)$. Well, this is simply a variant of the MLE fit in the previous section, except that now, instead of having labels $y_i = k$, we have soft labels, in the form of a weight on each sample. As we discussed before, we can fit a weighted classifier of this type easily simply by replacing all counts by weighted counts:

$$\mu_k = \frac{1}{\sum_{i=1}^{N} w_{ik}} \sum_{i=1}^{N} \mathbf{x}_i w_{ik}$$

$$\Sigma_k = \frac{1}{\sum_{i=1}^{N} w_{ik}} \sum_{i=1}^{N} (\mathbf{x}_i - \mu_k)(\mathbf{x}_i - \mu_k)^T w_{ik}.$$

We can also update our prior according to

$$p(y = k) = \frac{\sum_{i=1}^{N} w_{ik}}{\sum_{i=1}^{N} \sum_{k'=1}^{K} w_{ik'}}.$$

When we have a model and we obtain a label distribution $p(y_i = k|\mathbf{x}_i) = w_{ik}$, we are really getting a distribution over the *expected* labels (expected under our model), so we can call this the "expectation step" or E-step. When we have a distribution over labels and we wish to update our model parameters $p(y), \mu_1, \ldots, \mu_K, \Sigma_1, \ldots, \Sigma_K$, we are maximizing the likelihood of our expected labels, so we can call this the "maximization step," or M-step. If we simply alternate these two steps, we get an algorithm called the EM algorithm, which we can write like this:

Algorithm 1 EM clustering

```
Initialize means and covariances (more on this later)
while not converged do
    E-step: estimate w_ik for each datapoint i and each cluster k
    M-step: fit μ_k and Σ_k using the weighted MLE fit
end while
```

If this sounds familiar, it's because it is: this is just a "soft" version of K-means clustering. Instead of making a hard assignment of a label y_i to each datapoint $\mathbf{x}_i$, EM instead computes weights for each possible assignment based on the proximity of the corresponding cluster. Furthermore, EM allows each cluster to have not only a mean μ_k (which we previously denoted $\mathbf{c}_k$), but also a covariance Σ_k, which describes the shape of the cluster. This allows for large diffuse clusters and small dense ones, and the "nearest" cluster is no longer the one with the closest mean, but rather the one for which the mean is closest *under the corresponding covariance matrix*, so large clusters might still be "closest" even if they are further away in Euclidean space, if they have a larger covariance. We can modify EM to exactly recover K-means as following:

Algorithm 2 Hard EM clustering (k-means)

```
Initialize means and covariances (more on this later)
while not converged do
    E-step: estimate y_i = arg max_k w_ik for each datapoint i
    M-step: fit μ_k using the weighted MLE fit, set Σ_k = I
end while
```

4 EM Objective

Now we have a dataset, a hypothesis space (all Gaussian mixture models with K elements), and an algorithm (EM), but what is the objective? The objective of probabilistic clustering is marginal likelihood: the likelihood of the data if we *marginalize out* the unknown labels y_i – that is, the likelihood of the data if we average together the likelihoods for each possible assignment to y_i, weighted by

its probability. This likelihood is given by

$$\mathcal{L} = \prod_{i=1}^{N} p(\mathbf{x}_i) = \prod_{i=1}^{N} \sum_{k=1}^{K} p(y_i = k, \mathbf{x}_i) = \prod_{i=1}^{N} \sum_{k=1}^{K} p(y_i = k) p(\mathbf{x}_i | y_i = k). \quad (1)$$

It is in fact possible to optimize this likelihood directly, but it is very difficult, because there is no closed form solution. We can compute its gradient with respect to the parameters of the model (the entries in $p(y)$, the means μ_k, and the covariances Σ_k), and then use an algorithm like gradient ascent. However, this procedure is numerically tricky and extremely prone to poor local optima. The EM algorithm provides an efficient and simple method of optimizing this objective, by explicitly tracking the probability of each label for each datapoint, $p(y_i = k|\mathbf{x}_i) = w_{ik}$. We'll show this in detail in the next lecture, but for now we'll discuss the intuition. In EM, we consider a quantity called the *expected log likelihood*, which is given by

$$\hat{\mathcal{L}} = \sum_{i=1}^{N} \sum_{k=1}^{K} q(y_i = k|\mathbf{x}_i) \log p(y_i = k, \mathbf{x}_i) = \sum_{i=1}^{M} E_q[\log p(y_i = k, \mathbf{x}_i)],$$

where $q(y_i = k|\mathbf{x}_i)$ is the probability of points $\mathbf{x}_i$ having the label k given the model from the *previous* iteration of EM. That is, $q(y_i = k|\mathbf{x}_i) = w_{ik}$. The M-step directly optimizes the model parameters $p(y), mu_1, \ldots, \mu_K, \Sigma_1, \ldots, \Sigma_K$ with respect to the expected log-likelihood. The E-step doesn't change the model parameters, but it updates w_{ik} so that the distribution $q(y_i = k|\mathbf{x}_i)$ is equal to the latest model $p(y_i = k|\mathbf{x}_i)$: that is, it makes q more like p. So the intuition behind why EM works is that it alternates between optimizing a slightly different objective (the expected likelihood) in the M-step, and then modifying that objective to correspond to the current model p in the E-step. We will see a more formal connection to the marginal likelihood in the next lecture.