14 | Expectation Maximization

--

Learning Objectives:

- Explain the relationship between parameters and hidden variables.
- Construct generative stories for clustering and dimensionality reduction.
- Draw a graph explaining how EM works by constructing convex lower bounds.
- Implement EM for clustering with mixtures of Gaussians, and contrasting it with k-means.
- Evaluate the differences betweem EM and gradient descent for hidden variable models.

Dependencies:

Suppose you were building a naive Bayes model for a text categorization problem. After you were done, your boss told you that it became prohibitively expensive to obtain labeled data. You now have a probabilistic model that assumes access to labels, but you don't have any labels! Can you still do something?

Amazingly, you can. You can treat the labels as **hidden variables**, and attempt to learn them at the same time as you learn the parameters of your model. A very broad family of algorithms for solving problems just like this is the **expectation maximization** family. In this chapter, you will derive expectation maximization (EM) algorithms for clustering and dimensionality reduction, and then see why EM works.

14.1 Clustering with a Mixture of Gaussians

In Chapter 7, you learned about probabilitic models for classification based on density estimation. Let's start with a fairly simple classification model that *assumes* we have labeled data. We will shortly remove this assumption. Our model will state that we have K classes, and data from class k is drawn from a Gaussian with mean $\boldsymbol{\mu}_k$ and variance σ_k^2. The choice of classes is parameterized by $\boldsymbol{\theta}$. The generative story for this model is:

1. For each example $n = 1 \dots N$:
   (a) Choose a label $y_n \sim \mathcal{D}isc(\boldsymbol{\theta})$
   (b) Choose example $\boldsymbol{x}_n \sim \mathcal{N}or(\boldsymbol{\mu}_{y_n}, \sigma_{y_n}^2)$

This generative story can be directly translated into a likelihood as before:

$$p(D) = \prod_n \mathcal{M}ult(y_n \mid \boldsymbol{\theta}) \mathcal{N}or(\boldsymbol{x}_n \mid \boldsymbol{\mu}_{y_n}, \sigma_{y_n}^2) \tag{14.1}$$

$$= \overbrace{\prod_n \underbrace{\theta_{y_n}}_{\text{choose label}} \underbrace{\left[2\pi\sigma_{y_n}^2\right]^{-\frac{D}{2}} \exp\left[-\frac{1}{2\sigma_{y_n}^2}\left|\left|\boldsymbol{x}_n - \boldsymbol{\mu}_{y_n}\right|\right|^2\right]}_{\text{choose feature values}}}^{\text{for each example}} \tag{14.2}$$

If you had access to labels, this would be all well and good, and you could obtain closed form solutions for the maximum likelihood estimates of all parameters by taking a log and then taking gradients of the log likelihood:

$$\theta_k = \text{fraction of training examples in class } k \tag{14.3}$$
$$= \frac{1}{N}\sum_n [y_n = k]$$
$$\boldsymbol{\mu}_k = \text{mean of training examples in class } k \tag{14.4}$$
$$= \frac{\sum_n [y_n = k]\boldsymbol{x}_n}{\sum_n [y_n = k]}$$
$$\sigma_k^2 = \text{variance of training examples in class } k \tag{14.5}$$
$$= \frac{\sum_n [y_n = k]\,||\boldsymbol{x}_n - \mu_k||}{\sum_n [y_n = k]}$$

Suppose that you *don't* have labels. Analogously to the K-means algorithm, one potential solution is to iterate. You can start off with guesses for the values of the unknown variables, and then iteratively improve them over time. In K-means, the approach was the *assign* examples to labels (or clusters). This time, instead of making hard assignments ("example 10 belongs to cluster 4"), we'll make **soft assignments** ("example 10 belongs half to cluster 4, a quarter to cluster 2 and a quarter to cluster 5"). So as not to confuse ourselves too much, we'll introduce a new variable, $\boldsymbol{z}_n = \langle z_{n,1}, \ldots, z_{n,K}$ (that sums to one), to denote a fractional assignment of examples to clusters.

This notion of soft-assignments is visualized in Figure 14.1. Here, we've depicted each example as a pie chart, and it's coloring denotes the degree to which it's been assigned to each (of three) clusters. The size of the pie pieces correspond to the $\boldsymbol{z}_n$ values.

Formally, $z_{n,k}$ denotes the probability that example n is assigned to cluster k:

$$z_{n,k} = p(y_n = k \mid \boldsymbol{x}_n) \tag{14.6}$$
$$= \frac{p(y_n = k, \boldsymbol{x}_n)}{p(\boldsymbol{x}_n)} \tag{14.7}$$
$$= \frac{1}{Z_n}\mathcal{M}ult(k \mid \boldsymbol{\theta})\mathcal{N}or(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \sigma_k^2) \tag{14.8}$$

Here, the normalizer Z_n is to ensure that $\boldsymbol{z}_n$ sums to one.

Given a set of parameters (the $\boldsymbol{\theta}$s, $\boldsymbol{\mu}$s and σ^2s), the **fractional assignments** $z_{n,k}$ are easy to compute. Now, akin to K-means, given fractional assignments, you need to recompute estimates of the model parameters. In analogy to the maximum likelihood solution (Eqs (**??**)-(**??**)), you can do this by counting fractional points rather

? You should be able to derive the maximum likelihood solution results formally by now.

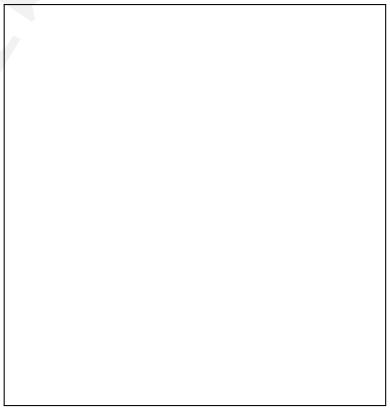

Figure 14.1: `em:piecharts`: A figure showing pie charts

Algorithm 37 GMM($\mathbf{X}$, K)

for $k = 1$ **to** K **do**
  $\boldsymbol{\mu}_k \leftarrow$ some random location // randomly initialize mean for kth cluster
  $\sigma_k^2 \leftarrow 1$ // initialize variances
  $\theta_k \leftarrow 1/K$ // each cluster equally likely a priori
end for
repeat
  for $n = 1$ **to** N **do**
    for $k = 1$ **to** K **do**
      $z_{n,k} \leftarrow \theta_k \left[2\pi\sigma_k^2\right]^{-\frac{D}{2}} \exp\left[-\frac{1}{2\sigma_k^2}||\boldsymbol{x}_n - \boldsymbol{\mu}_k||^2\right]$ // compute (unnormalized) fractional assignments
    end for
    $\boldsymbol{z}_n \leftarrow \frac{1}{\sum_k z_{n,k}} \boldsymbol{z}_n$ // normalize fractional assignments
  end for
  for $k = 1$ **to** K **do**
    $\theta_k \leftarrow \frac{1}{N}\sum_n z_{n,k}$ // re-estimate prior probability of cluster k
    $\boldsymbol{\mu}_k \leftarrow \frac{\sum_n z_{n,k}\boldsymbol{x}_n}{\sum_n z_{n,k}}$ // re-estimate mean of cluster k
    $\sigma_k^2 \leftarrow \frac{\sum_n z_{n,k}||\boldsymbol{x}_n - \mu_k||}{\sum_n z_{n,k}}$ // re-estimate variance of cluster k
  end for
until converged
return $\boldsymbol{z}$ // return cluster assignments

than full points. This gives the following re-estimation updates:

$$\theta_k = \text{fraction of training examples in class } k = \frac{1}{N}\sum_n z_{n,k} \tag{14.9}$$

$$\boldsymbol{\mu}_k = \text{mean of fractional examples in class } k = \frac{\sum_n z_{n,k}\boldsymbol{x}_n}{\sum_n z_{n,k}} \tag{14.10}$$

$$\sigma_k^2 = \text{variance of fractional examples in class } k = \frac{\sum_n z_{n,k}||\boldsymbol{x}_n - \mu_k||}{\sum_n z_{n,k}} \tag{14.11}$$

All that has happened here is that the hard assignments "$[y_n = k]$" have been replaced with soft assignments "$z_{n,k}$". As a bit of foreshadowing of what is to come, what we've done is essentially replace known labels with *expected labels*, hence the name "expectation maximization."

Putting this together yields Algorithm 14.1. This is the **GMM** ("**Gaussian Mixture Models**") algorithm, because the probabilitic model being learned describes a dataset as being drawn from a mixture distribution, where each component of this distribution is a Gaussian.

Just as in the K-means algorithm, this approach is succeptible to local optima and quality of initialization. The heuristics for comput-

? Aside from the fact that GMMs use soft assignments and K-means uses hard assignments, there are other differences between the two approaches. What are they?

ing better initializers for K-means are also useful here.

14.2 The Expectation Maximization Framework

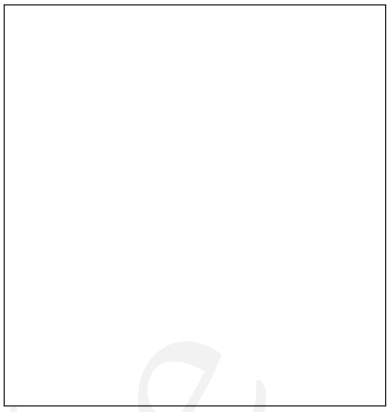

Figure 14.2: em:lowerbound: A figure showing successive lower bounds

At this point, you've seen a method for learning in a particular probabilistic model with hidden variables. Two questions remain: (1) can you apply this idea more generally and (2) why is it even a reasonable thing to do? Expectation maximization is a *family* of algorithms for performing maximum likelihood estimation in probabilistic models with hidden variables.

The general flavor of how we will proceed is as follows. We want to maximize the log likelihood $\mathcal{L}$, but this will turn out to be difficult to do directly. Instead, we'll pick a surrogate function $\tilde{\mathcal{L}}$ that's a lower bound on $\mathcal{L}$ (i.e., $\tilde{\mathcal{L}} \leq \mathcal{L}$ everywhere) that's (hopefully) easier to maximize. We'll construct the surrogate in such a way that increasing it will force the true likelihood to also go up. After maximizing $\tilde{\mathcal{L}}$, we'll construct a *new* lower bound and optimize that. This process is shown pictorially in Figure 14.2.

To proceed, consider an arbitrary probabilistic model $p(\boldsymbol{x}, \boldsymbol{y} \mid \boldsymbol{\theta})$, where $\boldsymbol{x}$ denotes the observed data, $\boldsymbol{y}$ denotes the hidden data and $\boldsymbol{\theta}$ denotes the parameters. In the case of Gaussian Mixture Models, $\boldsymbol{x}$ was the data points, $\boldsymbol{y}$ was the (unknown) labels and $\boldsymbol{\theta}$ included the cluster prior probabilities, the cluster means and the cluster variances. Now, given access *only* to a number of examples $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, you would like to estimate the parameters ($\boldsymbol{\theta}$) of the model.

Probabilistically, this means that some of the variables are unknown and therefore you need to marginalize (or sum) over their possible values. Now, your data consists only of $\mathbf{X} = \langle \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_N \rangle$, not the $(\boldsymbol{x}, y)$ pairs in D. You can then write the likelihood as:

$$p(\mathbf{X} \mid \boldsymbol{\theta}) = \sum_{y_1} \sum_{y_2} \cdots \sum_{y_N} p(\mathbf{X}, y_1, y_2, \ldots y_N \mid \boldsymbol{\theta}) \qquad \text{marginalization} \tag{14.12}$$

$$= \sum_{y_1} \sum_{y_2} \cdots \sum_{y_N} \prod_n p(\boldsymbol{x}_n, y_n \mid \boldsymbol{\theta}) \qquad \text{examples are independent} \tag{14.13}$$

$$= \prod_n \sum_{y_n} p(\boldsymbol{x}_n, y_n \mid \boldsymbol{\theta}) \qquad \text{algebra} \tag{14.14}$$

At this point, the natural thing to do is to take logs and then start taking gradients. However, once you start taking logs, you run into a problem: the log cannot eat the sum!

$$\mathcal{L}(\mathbf{X} \mid \boldsymbol{\theta}) = \sum_n \log \sum_{y_n} p(\boldsymbol{x}_n, y_n \mid \boldsymbol{\theta}) \tag{14.15}$$

Namely, the log gets "stuck" outside the sum and cannot move in to decompose the rest of the likelihood term!

The next step is to apply the somewhat strange, but strangely useful, trick of multiplying by 1. In particular, let $q(\cdot)$ be an arbitrary probability distribution. We will multiply the $p(\dots)$ term above by $q(y_n)/q(y_n)$, a valid step so long as q is never zero. This leads to:

$$\mathcal{L}(\mathbf{X} \mid \boldsymbol{\theta}) = \sum_n \log \sum_{y_n} q(y_n) \frac{p(\boldsymbol{x}_n, y_n \mid \boldsymbol{\theta})}{q(y_n)} \tag{14.16}$$

We will now construct a lower bound using **Jensen's inequality**. This is a very useful (and easy to prove!) result that states that $f(\sum_i \lambda_i x_i) \geq \sum_i \lambda_i f(x_i)$, so long as (a) $\lambda_i \geq 0$ for all i, (b) $\sum_i \lambda_i = 1$, and (c) f is concave. If this looks familiar, that's just because it's a direct result of the definition of **concavity**. Recall that f is concave if $f(ax + by) \geq af(x) + bf(x)$ whenever $a + b = 1$.

? Prove Jensen's inequality using the definition of concavity and induction.

You can now apply Jensen's inequality to the log likelihood by identifying the list of $q(y_n)$s as the λs, log as f (which is, indeed, concave) and each "x" as the p/q term. This yields:

$$\begin{aligned}
\mathcal{L}(\mathbf{X} \mid \boldsymbol{\theta}) &\geq \sum_n \sum_{y_n} q(y_n) \log \frac{p(\boldsymbol{x}_n, y_n \mid \boldsymbol{\theta})}{q(y_n)} && (14.17)\\
&= \sum_n \sum_{y_n} \Big[q(y_n) \log p(\boldsymbol{x}_n, y_n \mid \boldsymbol{\theta}) - q(y_n) \log q(y_n) \Big] && (14.18)\\
&\triangleq \tilde{\mathcal{L}}(\mathbf{X} \mid \boldsymbol{\theta}) && (14.19)
\end{aligned}$$

Note that this inequality holds for *any* choice of function q, so long as its non-negative and sums to one. In particular, it needn't even by the same function q for each n. We will need to take advantage of both of these properties.

We have succeeded in our first goal: constructing a lower bound on $\mathcal{L}$. When you go to optimize this lower bound for $\boldsymbol{\theta}$, the only part that matters is the first term. The second term, $q \log q$, drops out as a function of $\boldsymbol{\theta}$. This means that the the maximization you need to be able to compute, for fixed q_ns, is:

$$\boldsymbol{\theta}^{(\text{new})} \leftarrow \arg\max_{\boldsymbol{\theta}} \sum_n \sum_{y_n} q_n(y_n) \log p(\boldsymbol{x}_n, y_n \mid \boldsymbol{\theta}) \tag{14.20}$$

This is *exactly* the sort of maximization done for Gaussian mixture models when we recomputed new means, variances and cluster prior probabilities.

The second question is: what should $q_n(\cdot)$ actually be? Any reasonable q will lead to a lower bound, so in order to choose one q over another, we need another criterion. Recall that we are hoping to maximize $\mathcal{L}$ by instead maximizing a lower bound. In order to ensure that an increase in the lower bound implies an increase in $\mathcal{L}$, we need

to ensure that $\mathcal{L}(\mathbf{X} \mid \boldsymbol{\theta}) = \tilde{\mathcal{L}}(\mathbf{X} \mid \boldsymbol{\theta})$. In words: $\tilde{\mathcal{L}}$ should be a lower bound on $\mathcal{L}$ that makes contact at the current point, $\boldsymbol{\theta}$. This is shown in Figure **??**, including a case where the lower bound does *not* make contact, and thereby does not guarantee an increase in $\mathcal{L}$ with an increase in $\tilde{\mathcal{L}}$.

14.3 EM versus Gradient Descent

computing gradients through marginals
  step size

14.4 Dimensionality Reduction with Probabilistic PCA

derivation
  advantages over pca

14.5 Exercises

Exercise 14.1. TODO...