Gaussian Mixture Models and Expectation Maximization

Tinne Tuytelaars
Luc Van Gool

Background literature:
https://www.ee.washington.edu/techsite/papers/documents/UWEETR-2010-0002.pdf

So far, we mostly worked with normal distributions. However, not all data follow a normal distribution.
In this chapter, we will introduce Gaussian Mixture Models (GMM), which can be used to approximate *any* distribution.
However, this comes at a cost: fitting a GMM to a set of data points is not as simple as fitting a normal distribution.
To solve this, we will need the EM algorithm, which stands for Expectation Maximization.
Note though that EM is a general technique, that can be used in lots of different situations, not just for fitting GMMs.

If something in the slides is unclear or you want to delve deeper in the topic of EM, follow the link mentioned in the slide.

Overview

- Gaussian Mixture Model (GMM)
  - A flexible model that can approximate a wide range of distributions
- Expectation Maximization
  - An iterative scheme for solving a Maximum Likelihood problem with hidden variables, that can be used e.g. to fit a GMM to a set of datapoints

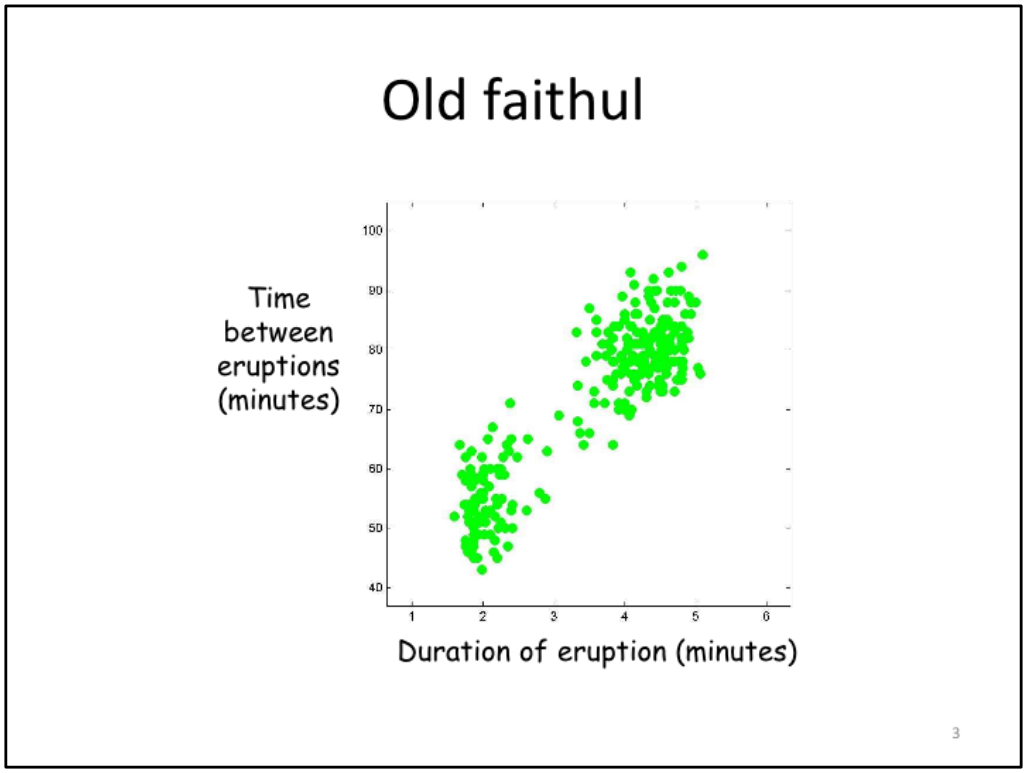
1

In general, we can say that EM is a method to solve Maximum Likelihood problems that involve hidden variables.

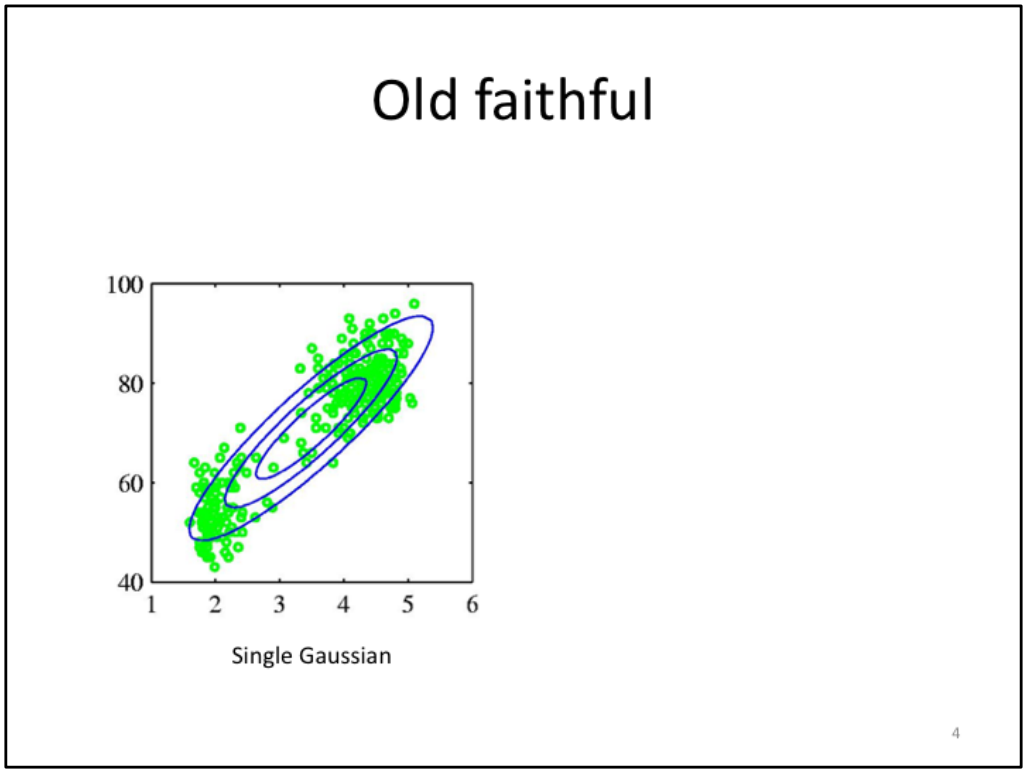
Example: Old faithful

2

Let's start with a motivation for GMMs, using data collected from a geyser in Yellowstone National Park known as 'Old Faithful'.
It's called that way, because it erupts roughly ones every hour to an hour and a half.
So let us look at some data collected from those eruptions – in particular the duration of an eruption, and the time in between two eruptions.



Looking at this data, it's clear that the two measurements are not independent: they are in fact strongly correlated:
the longer an eruption lasts, the longer one will have to wait for the next eruption.



So what do we get if we would try to fit a normal distribution to this data ?
Based on what we have seen before, we can easily estimate the parameters of the normal distribution: the mean and covariance matrix, using MLE:
we write down p(z|mu, sigma^2) and find the parameters mu and sigma that maximize this expression.
The result is shown in blue in the figure above, using iso-probability contours.

However, this model does not match the actual data very well. ~~: sampling from this normal distribution, one would expect 67% of the data to fall within the inner ellipse, corresponding to the region within one standard devation from the mean. Likewise, 95% of the data should fall within the middle ellipse, and 99% within the outer ellipse. This is not what we observe in practice…~~

In fact, the data is more bimodal in nature: there seem to be two clusters within the data: a lot of eruptions last for 2 minutes, and a lot of them last for 4-5 minutes. Fitting a Gaussian to each of these two groups, as shown in the figure on the right, results in a much more accurate model. This is what we get if we fit a GMM to the data.

Mixtures of Gaussians

Mixture model:

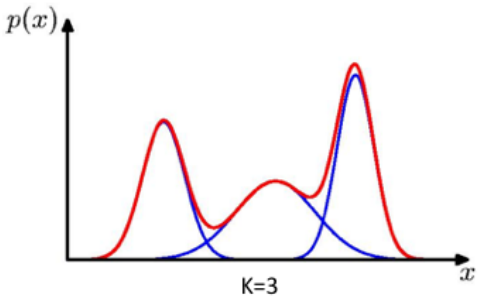
Combine simple models into a complex model

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

Component

Mixing coefficient

$$\forall k : \pi_k \geqslant 0 \qquad \sum_{k=1}^{K} \pi_k = 1$$



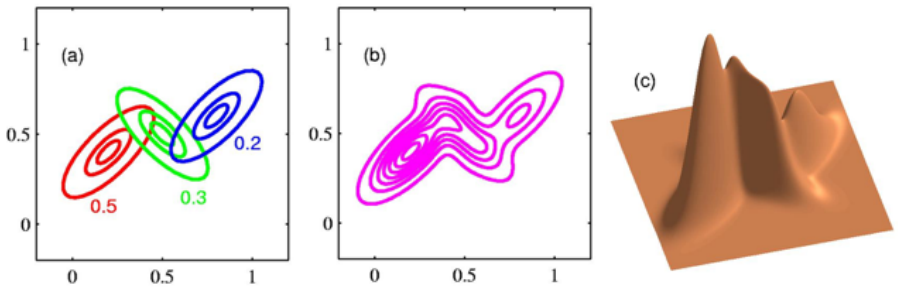
5

A mixture model is a model that's composed of a number of simpler models or components, that are weighed, then summed together.
A Gaussian mixture model is a model where the components are Gaussians (normal distributions).

The weights are referred to as 'mixing coefficients'. They're always bigger or equal to zero, and they sum together to 1.
This way, it's guaranteed that the mixture model fulfill the conditions of a probability distribution, if the components are also probability distributions.

Mixture of Gaussians

Flexible model that can approximate various distributions



6

A GMM is a very flexible model: it can be used when we have clearly distinctive groups in our data, as was the case in the example of Old Faithful.
However, also if this is not the case, a distributions can be approximated well using a GMM. The distribution shown on the right is clearly not normally distributed, and does not consist of different groups or clusters that are normally distributed, yet it is obtained as a GMM.

Mixture of Gaussians

- GMM are universal approximators of densities (as long as there are sufficient Gaussians)
- Full rank GMMs are not easy to deal with (too many parameters: nb of dimensions squared)
- Even diagonal GMMs are universal approximators

7

In fact, one can prove that GMMs are universal approximators of probability densities: they can approximate *any* pdf, no matter what it looks like, as long as the number of components is chosen sufficiently high.

When working in high dimensions, the number of parameters of a GMM quickly becomes too large to handle: given K components in a N-dimensional space, we have
- K mixture components
- K N-dimensional vectors for the mean
- K NxN-dimensional matrices for the covariance.

Especially the latter are expensive. One needs a lot of data to accurately estimate all these parameters.

To alleviate this problem, a simpler model is often used instead when working in high-dimensional spaces, where the components are normal distributions with diagonal covariance matrices.
In that case, the number of parameters is O(N) instead of O(N^2): K mixture components + K N-dimensional vectors for the mean + K N-dimensional diagonal matrices.
It can be proven that, even when using diagonal GMMs, they're still universal approximators (but you will need a higher number of components than before).

Mixture of Gaussians

- Mixture coefficients can be interpreted as prior probabilities:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

$$p(\mathbf{x}) = \sum_{k=1}^{K} p(k)p(\mathbf{x}|k)$$

Note the similarity between the equation for GMM and the product rule of probability theory.
Based on this, one can give a probabilistic interpretation to GMMs, where the mixture coefficients correspond to prior probabilities of the different components and the different components represent the conditional probabilities of the data given the selected component.

To make the above clearer, think of a game of darts with three players, and your task is to guess where the arrow will end up on the board.
It seems reasonable to assume that for a given player, this follows a normal distribution (with larger variance for an inexperienced player).
That's p(x|k), where x is the position on the board and k is an index specifying the player.
But what if you don't know who threw the arrow ? In that case, you can still estimate the distribution, as follows:
p(x) = sum(p(k). p(x|k)) = sum(p(k). N(x| m_k, S_k))
… and this turns out to be a GMM !

Sampling from the GMM

- To generate a data point:
  - First pick one of the components with probability π_k
  - Then draw a sample x_n from that component
- Repeat these two steps for each data point

9

This corresponds to a generative model, where data are generated as follows:
- first, a component is sampled, based on the prior probabilities p(k).
- then, a datapoint is sampled using the component-specific data distribution p(x|k).

This process of sampling data from the model is relatively easy to implement. However, the difficult part is the inverse process: given the data, infer the model (i.e. the parameters of the GMM).

Fitting the GMM

- In practice, we mostly wish to solve the inverse process: given a data set, find the corresponding parameters
- Parameters involved:
  - Mixing coefficients
  - Means
  - Covariances
- Solvable with Max. Likelihood estimation ?

10

Fitting a GMM model to a set of data points involves estimating all the parameters involved:
- The mixing coefficients (K scalars, with K the number of components in the mixture model)
- The means of the different components (K D-dimensional vectors, where D is the dimensionality of the data)
- The covariances of the different components (K DxD-dimensional matrices)

So can we use Maximum Likelihood estimation to find these parameters ? Let us try …

Fitting the GMM

- Determining parameters using maximum likelihood (or max. log-likelihood)

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

Log of a sum; no closed form maximum.

- Solution:
  - use standard, iterative, numeric optimization methods, or
  - Use the *expectation maximization* algorithm (see below).

11

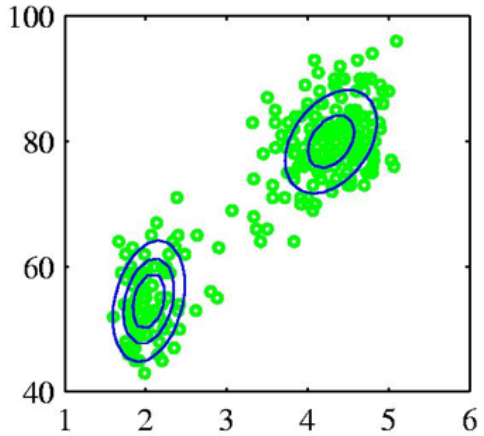
The likelihood is the conditional probability of the data (= all N data samples) , given the parameters of the model.
Deriving the formula in the slide is left as an exercise. Basically, you follow these steps:
- Use the fact that all N data samples are sampled from the GMM independently.
- Take the logarithm of both the left and right hand side.
- Use the fact that the logarithm of a product is equal to the sum of the logarithms.

Finding the maximum likelihood solution is equal to finding the maximum of the loglikelihood, i.e. the expression in the slide. This could be done by taking the derivative of the expression and setting that equal to zero. Unfortunately, the math is not as simple as what we had earlier in the case of a normal distribution. The logarithm of the sum in the expression prevents a simple closed form solution. Instead, one has to fall back to standard iterative numeric optimization methods. Or, alternatively, use the Expectation Maximization algorithm described below (which provides more insight than the numerical solution).

We will first explain the basic idea of Expectation Maximization (or EM for short) for the example of fitting a GMM model. However, EM is much more generic than that. It's a method that can be applied to solve a much wider set of problems, as we will discuss later.

Basic idea

- What if we knew which component generated each data point ?



Ok, let us first explain the basic idea, for this specific case of fitting a GM model. Here's a first insight: suppose we knew, for each data point, a 'label' indicating which component of the distribution it was sampled from. Then finding the GMM parameters would be simple: take all the points labeled as coming from the first component and fit a normal distribution to it, then switch to the second component, and so on.

Fitting the GMM

- If we knew which component generated each data point, we could avoid the sum inside the logarithm, and the maximum likelihood solution would simply involve fitting each component to the corresponding set of points.
- Problem: we don't know these 'labels'
- We shall refer to the labels as *latent* (= hidden) variables

13

Unfortunately, we do not know which component generated a data point. We refer to these 'labels' as missing data or hidden variables.
Hidden variables are additional variables, that are unknown. We're not really interested in them per se, yet they are useful as auxiliary variables, as they make it easier to solve the problem at hand. We refer to them as 'unobserved', 'hidden' or 'latent' variables, as opposed to the regular 'observed' variables (the measured data) or 'unknowns' (in this case: the parameters of the GMM).

Expectation-Maximization

= an iterative method to estimate the maximum likelihood solution of a probability density function when not all data are observed / dealing with *hidden* variables

- Observed data y
- Complete data x (= observed + unobserved)

(sometimes hidden variables are 'invented' to make problem easier to solve)

14

Expectation-Maximization is a method to solve maximum likelihood problems with hidden variables. It's an iterative method, that alternates between updating the estimate on the hidden variables and updating the estimates of the unknowns, until convergence.

In the following slides, we use the following convention: y denotes the observed data, i.e. the measured data points, while x denotes all the data (both observed and unobserved).

Expectation-Maximization

- Maximum Likelihood estimation:

$$\hat{\theta}_{\mathrm{MLE}} = \arg \max_{\theta \in \Omega} p(y \mid \theta).$$

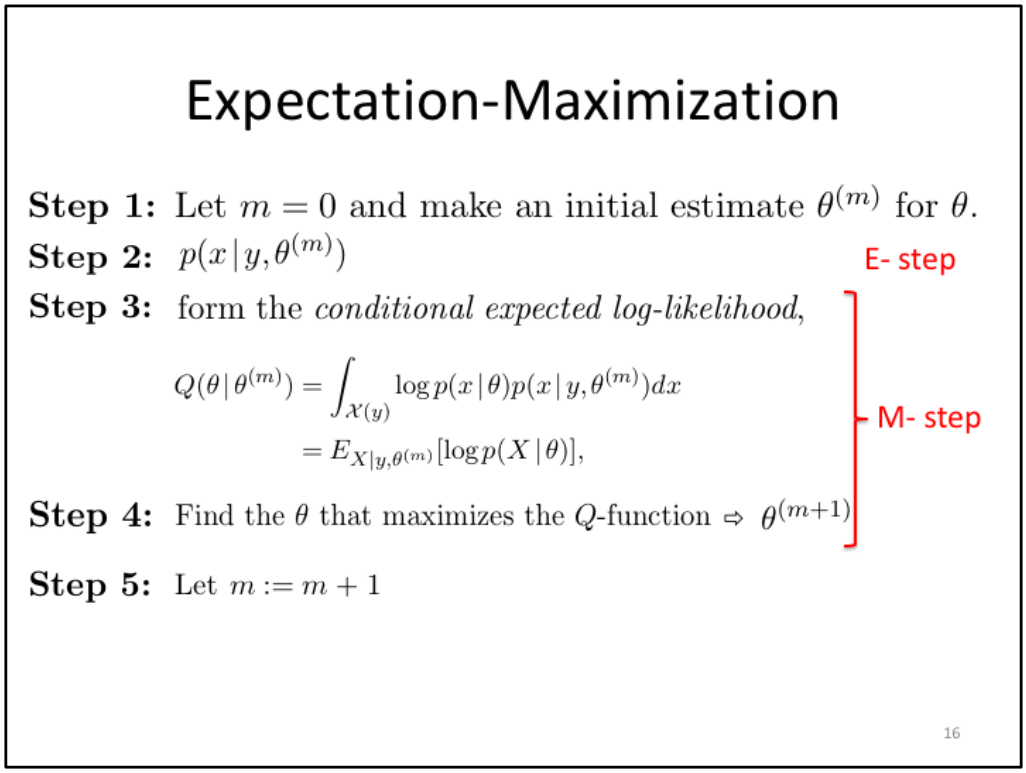
- *Intuitively*, this consists of iterating over two steps:
  - Make a guess about the complete data X. 'E- step'
  - Find the solution that maximizes the likelihood of X. 'M- step'
- *To be precise*, instead of 'guessing' x, we work with $p(x \mid y, \theta^{(m)})$ using the params estimated in the previous iteration.
- Instead of the MLE, we then use *the expected value* of the MLE.

15

EM exploits the fact that the hidden variables have been chosen such that, once they are known, it becomes relatively easy to solve the problem at hand: maximizing the likelihood of the data.
So it iterates over two steps: an "Expectation step" or "E-step", and a "Maximization step" or "M-step".
Roughly speaking, during the E-step, the estimate of the hidden variables is updated, given the model estimated in the previous M-step; and during the M-step the likelihood is maximized, given the estimate for the hidden variables from the previous E-step.

More precisely, we deal with the uncertainty on the hidden variables, and so in the E-step, we do not just make a guess, but rather compute the probability distribution over the hidden variables, based on the model parameters from the previous iteration. Then during the M-step, we **maximize the expected value of the likelihood**, using the probability distribution over the hidden variables of the previous step.

The algorithm iterates over these two steps, until convergence.



Here's a more detailed overview of the whole algorithm.

Step 1 is the initialization. Variable m is an index counting the number of iterations. Additionally, we also need an initial estimate for the parameters of our model to start with. In practice, these are often chosen at random or based on a prior.
Step 2 estimates the probability distribution of the complete data, given the observed data and the parameters from the previous iteration. This is the E-step.
Step 3 and 4 then continue to estimate new parameters theta, based on the idea of maximum likelihood estimation. We forget about the theta from the previous iteration, only keeping the probability distribution of x from the previous step.
In step 3, we want to estimate the log-likelihood, log p(x | θ) . But since we do not really know x, we'll have to maximize the *expected* log p(x | θ) instead. So, we will integrate over all possible values of x, and for each possible value of x, we weight log p(x | θ) by the *probability of seeing that* x. However, we don't really know the probability of seeing each x, all we have is the guess that we made in Step 2, which was p(x | y, θ(m)), so we will use that as an approximation. We refer to this function to estimate the expected log likelihood as the Q-function.
Step 4 then determines a new estimate for the parameters theta by selecting the parameters that maximize the Q-function of step 3.
Step 5 increments m to keep track of the number of iterations. And then we go back to step 2.

Expectation-Maximization

- Auxiliary objective function

$$Q(\theta \mid \theta^{(m)}) = \int_{\mathcal{X}(y)} \log p(x \mid \theta) p(x \mid y, \theta^{(m)}) dx$$
$$= E_{X \mid y, \theta^{(m)}}[\log p(X \mid \theta)],$$

- In short, EM iterates over

$$\theta^{j+1} = \arg\max_{\theta^{j+1}} Q(\theta^j, \theta^{j+1})$$

- Warning: actual implementations of EM can become quite messy (if using complex models)

17

Note how this auxiliary objective function Q is what really matters: it gives a relation between the previous estimate of the parameters and the new estimate.
When implementing, you do not need to separate E and M steps: simply apply the argmax-equation above over and over again, until convergence.
However, in order to derive Q for a given problem, it's good to do it step-by-step, writing down equations for the E and M steps separately, as the math can become quite complicated (depending on the models used).

Expectation-Maximization

- Under some mild conditions, EM is proved to converge (likelihood increases in each iteration), but not necessarily to global optimum.
- Therefore, it may be a good idea to run EM a few times with different random initializations, then pick the best solution.

18

Under some mild conditions, it can be shown that the EM algorithm converges to a stable solution (in each iteration, the EM estimate is guaranteed to never get worse*)*. However, the problem is non-convex, so there's no guarantee that the solution found corresponds to the global optimum.
In practice, it is common to start EM from multiple random initializations, and choose the one with the largest likelihood as the final guess for θ.

EM for GMMs

- Parameters:
  - Mixing coefficients: p_k
  - Means: μ_k
  - Covariances: Σ_k } θ_k
- Measurements / Observed variables:
  - Data points: x_i
- Latent variables:
  - Component generating x: z_{ik}
  - $p(z_{ik} | x_i, \theta^m) = \gamma_{ik}$

Different iterations of these variables indicated with superscripts.

19

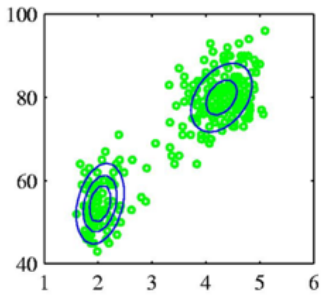
Ok, so let us now apply the above theory to our original problem of fitting a GMM model to a set of data points.
We rename the parameters as indicated above. The mean and covariance of component k are grouped together in theta_k.
As indicated earlier, the latent variables in this case are a set of 'labels' that indicate, for each data point, which component of the mixture model they belong to. To keep equations clean, we use indicator vectors for this: z_{ik} (i.e., one vector for each data point i, where the k-th element, z_{ik}, indicates whether data point i belongs to component k or not).

EM for GMMs

the contribution of training sample i to Gaussian k

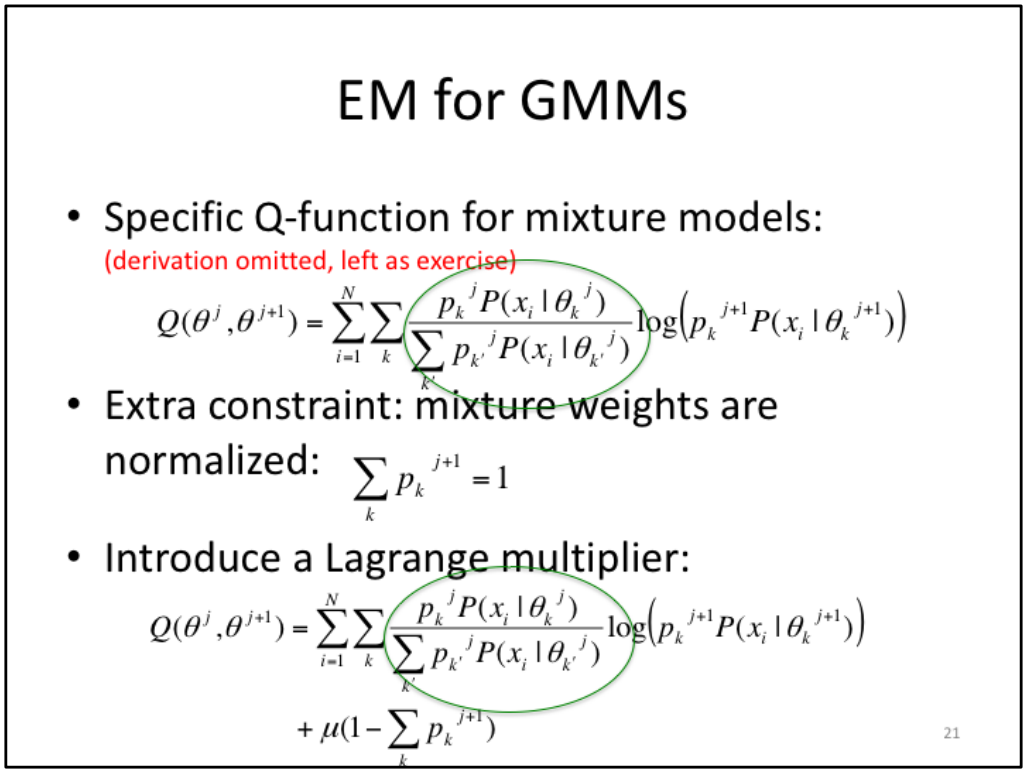
$p(z_{ik} \mid x_i, \theta^m) = \dfrac{p_k{}^j P(x_i \mid \theta_k{}^j)}{\sum_{k'} p_{k'}{}^j P(x_i \mid \theta_{k'}{}^j)}$ γ_{ik}

$$Q(\theta \mid \theta^{(m)}) = \int_{\mathcal{X}(y)} \log p(x \mid \theta) p(x \mid y, \theta^{(m)}) dx$$
$$= E_{X \mid y, \theta^{(m)}}[\log p(X \mid \theta)],$$



To apply EM, we first need to have an expression for p(x|y,theta) (step 2 from a few slides above). This is the conditional probability distribution for the complete data, given the observed data and the model parameters from a previous iteration. The complete data is both the observed as well as the unobserved data, but since the observed data is fixed, we only need to focus on the hidden variables: the 'labels' for each data point. For a particular datapoint i, this probability can be written as the equation on top of the slide (by applying Bayes' rule). We refer to this expression as γ_{ik}.

Next (step 3 a few slides above), we need to find the expression for the Q-funtion. To this end, we fill in the equation for p(x|y,theta).



After a few more steps, we get the expression for the Q-function shown here (full derivation left as an exercise).
The first summation runs over all data points. The second summation runs over all components.
The expression within the logarithm only involves one component, as this likelihood builds on the knowledge of the hidden variables, which indicates which component to use. The factor in front of the logarithm makes sure that only those data points belonging to the component contribute (or to be more precise: data points only contribute with a weight equal to their probability of belonging to that particular component).

There's one extra constraint on the parameters of the GMM we need to bring into account, namely the fact that the sum of all the mixture coefficients is equal to one. This gives us a constrained optimization problem, which can be solved by introducing a Lagrange multiplier.

To find the maximum of the Q function, we need to take the derivatives wrt to the parameters as well as wrt the Lagrange multiplier.

Not familiar with the method of Lagrange multipliers? This online tutorial can help you out: http://www.slimy.com/~steuard/teaching/tutorials/Lagrange.html

EM for GMMs

- Taking first derivative to find maximum:

$$\frac{\partial \hat{Q}(\theta^j, \theta^{j+1})}{\partial p_k^{\,j+1}} = \sum_{i=1}^{N} \frac{p_k^{\,j} P(x_i \mid \theta_k^{\,j})}{\sum_{k'} p_{k'}^{\,j} P(x_i \mid \theta_{k'}^{\,j})} \frac{1}{p_k^{\,j+1}} - \mu = 0$$

- Solve for new mixture weight:

$$p_k^{\,j+1} = \frac{1}{\mu} \sum_{i=1}^{N} \frac{p_k^{\,j} P(x_i \mid \theta_k^{\,j})}{\sum_{k'} p_{k'}^{\,j} P(x_i \mid \theta_{k'}^{\,j})}$$

Still to be determined…

22

Taking the derivative wrt the mixture coefficients gives the expression on top. Setting this derivative equal to zero and solving for the mixture coefficient, we get the expression below. This expression gives the new value for the mixture coefficient for the k-th component of our GMM as a function of the parameters from the previous iteration and the still unknown Lagrange multiplier.

EM for GMMs

- $$1 = \sum_k p_k^{j+1}$$ Normalization

$$= \frac{1}{\mu} \sum_{i=1}^{N} \frac{\sum_k p_k^j P(x_i \mid \theta_k^j)}{\sum_{k'} p_{k'}^j P(x_i \mid \theta_{k'}^j)}$$ Plug in result from optimization (previous slide)

$$= \frac{1}{\mu} \sum_{i=1}^{N} 1 = \frac{N}{\mu}$$

$$\Rightarrow \quad \mu = N$$

23

But we know all the mixture coefficients should sum to one (this is also what you get if you would take the derivative wrt the Lagrange multiplier).
Filling in the expression from the previous slide and rewriting this expression, we find the Lagrange multiplier mu is equal to the total number of data points.

EM for GMMs

$$p_k^{\,j+1} = \frac{1}{N}\sum_{i=1}^{N}\frac{p_k^{\,j} P(x_i \mid \theta_k^{\,j})}{\sum_k p_{k'}^{\,j} P(x_i \mid \theta_{k'}^{\,j})} = \gamma_{ik}$$

In words: new mixture coefficient for component k is computed by

- Computing the probability of each data point x_i belonging to component k under the model of the previous iteration.
- Taking the average over all data points.

24

Replacing the Lagrange multiplier mu by the total number of points N in the expression for the new values of the mixture coefficients we derived two slides back, gives us the expression above. The term encircled in green is the probability of datapoint i belonging to component k, under the model of the previous iteration. The new value for the mixture coefficients is the average of this probability over all datapoints. In other words: components that do a good job in explaining the data will gain in importance, while components that do not explain the data well will get smaller weights.

EM for GMMs

- Next, determine the other parameters:

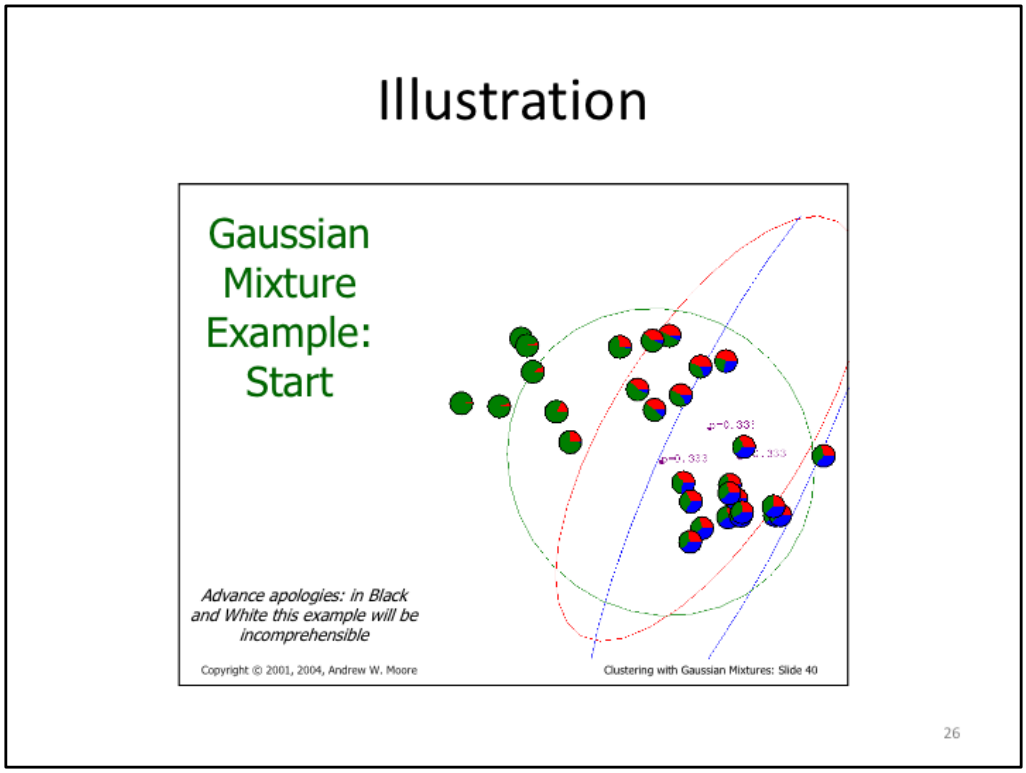
Means: $$\mu_k^{j+1} = \frac{1}{\sum_{i=1}^{N} \gamma^{j}_{i,k}} \sum_{i=1}^{N} \gamma^{j}_{i,k} x_i$$

Covariances: $$\Sigma_k^{j+1} = \frac{1}{\sum_{i=1}^{N} \gamma^{j}_{i,k}} \sum_{i=1}^{N} \gamma^{j}_{i,k} (x_i - \mu_k)(x_i - \mu_k)^t$$

- In words: fit a gaussian to the data, with data points being weighted with $\gamma^{j}_{i,k}$

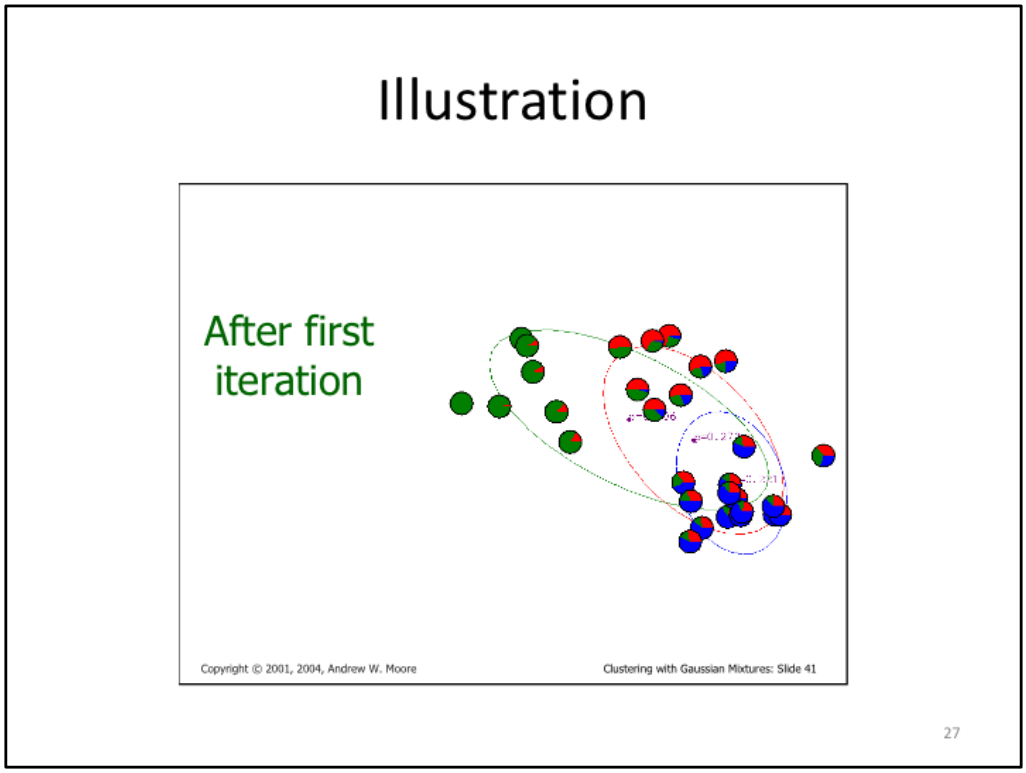
25

Ok, so we now have an expression to find updated values for the mixture coefficients. What about the other parameters ?
Following a similar derivation, now taking derivatives wrt θ_k^{j+1}, we obtain the expressions shown above.
As follows from these equations, these are relatively simple to estimate based on the γ_{ik} (i.e. the probability distribution over the hidden variables): one simply has to fit a normal distribution to the data, with each data point weighted by γ_{ik}.



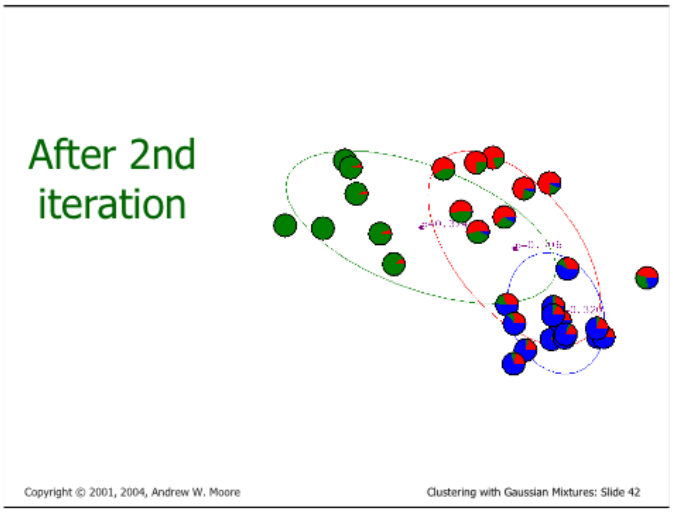
Here's a visual illustration of a particular example, where the goal is to fit a GMM model with 3 components to a set of datapoints.
One component is shown in red, one in blue, one in green. Ellipses represent an iso-probability curve of a given component based on the current estimate of the parameters.
For each datapoint, a small disc pie chart shows the conditional probability distribution over the hidden variables, given the current estimate of the model parameters.

Here's the start situation: the 3 GMM components have been initialized with random parameters and do not really fit the data.
The pie charts for the data show the probability that a specific datapoint belongs to any of the three components (E-step).
Based on these conditional probability distributions of the hidden variables, we then move on to the M-step: re-estimating the model parameters by maximizing the expected log-likelihood. As we have seen above, this boils down to fitting a normal distribution to the data, using the conditional probabilities as weights.



Here you see what this gives. The models already better fit the data.
Based on the new model parameters, the conditional probabilities of the hidden variables(pie-charts) are re-estimated.

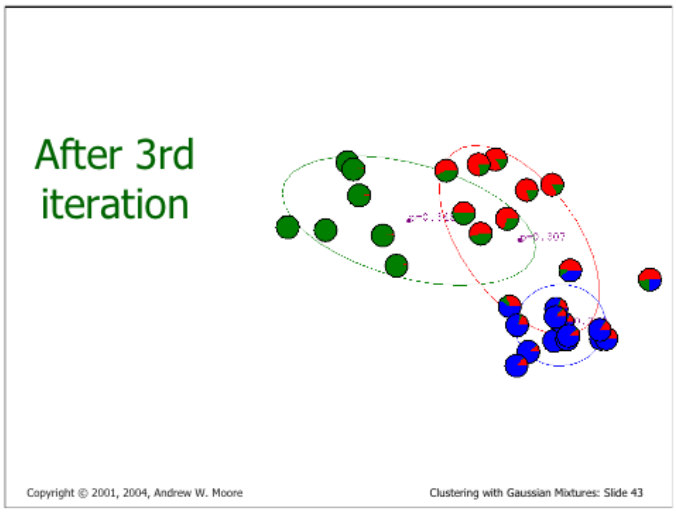
Illustration



28

Same for the next iteration

Illustration

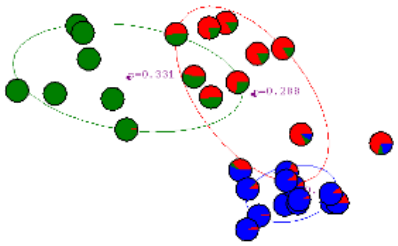


29

… and the next

Illustration

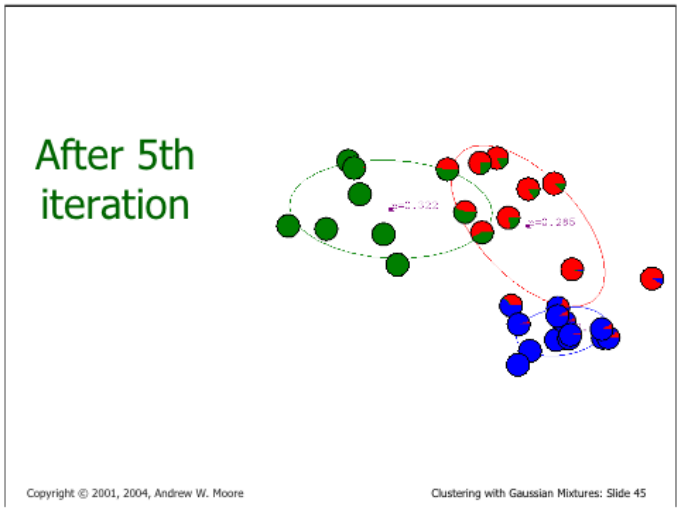
After 4th iteration



Clustering with Gaussian Mixtures: Slide 44

30

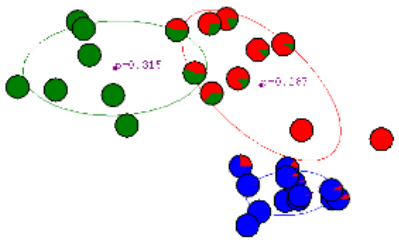
Illustration



31

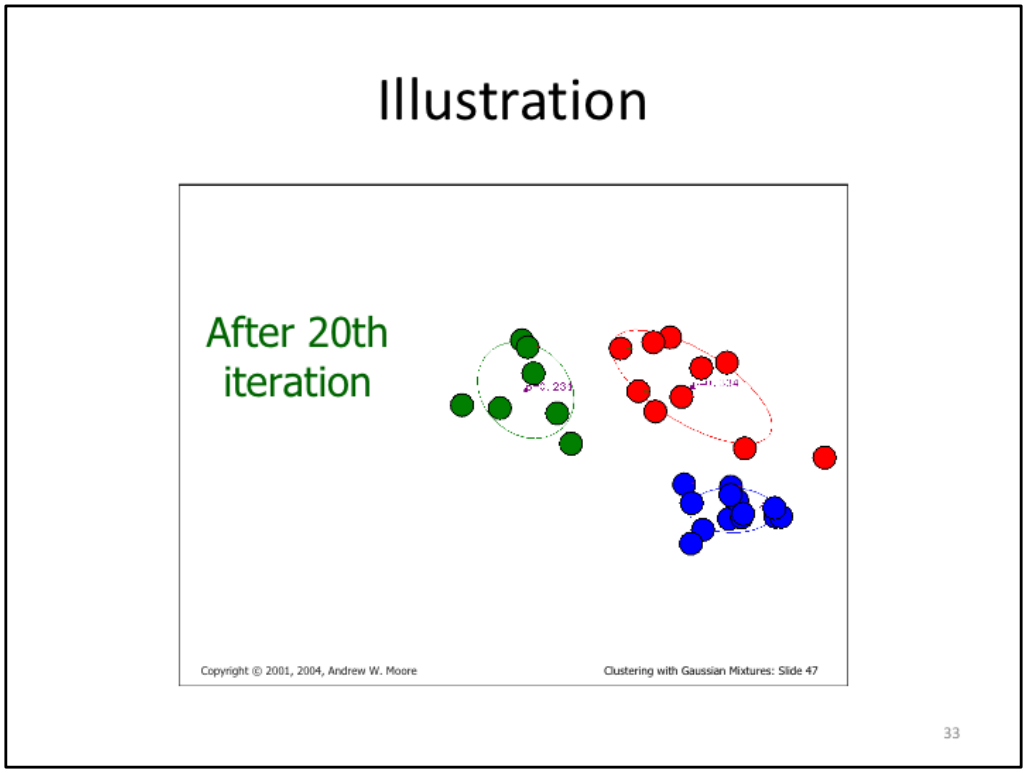
Illustration

After 6th iteration



Clustering with Gaussian Mixtures: Slide 46

32



After a few more iterations, the situation does not change much anymore and we're close to convergence.
Note how, in this case, all data points are almost exclusively assigned to a single component. This happens when the data consists of clearly distinguishable clusters, but is not necessarily the case when fitting GMM models to data in general.

Singularity problem

- When number of observations is not very large compared to number of Gaussian components, you may get singularity problems (Gaussian fit to a single data point will get variance of 0).
- This is in fact a general problem of MLE
- Solution: - apply a heuristic

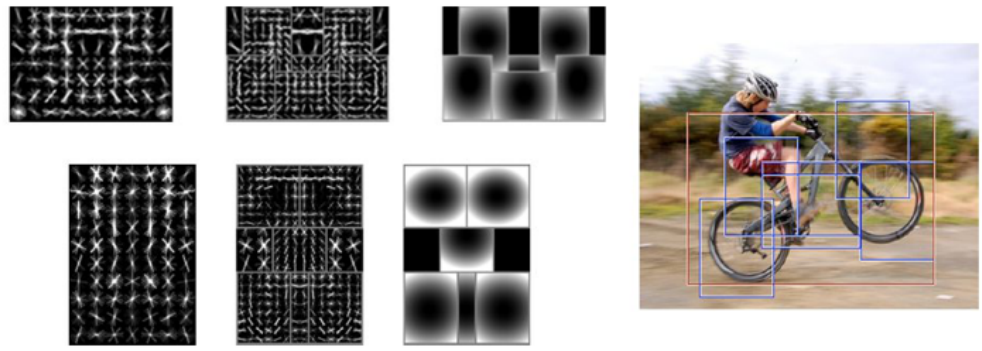
  (e.g. remove component,
  or re-initialize that component with random values)

  - MAP- EM

    (i.e. add extra term with prior probabilities for parameters)

34

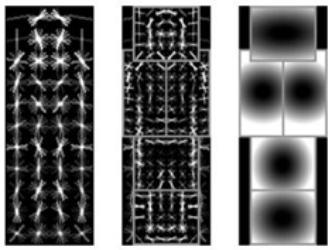
Discriminative part-based models



P. Felzenszwalb, R. Girshick, D. McAllester, D. Ramanan, Object Detection with Discriminatively Trained Part Based Models, PAMI 32(9), 2010

An example of a practical (albeit somewhat outdated) system relying on expectation maximization and latent variables in computer vision
was the Deformable Parts Model by Felzenszwalb et al.
There, an object detector was built that consisted of both a 'root filter', covering the entire object, as well as several 'part filters', that focused on specific parts of the object. To allow for changes in pose, within-class variability, etc., the parts were not at fixed locations relative to the object's bounding box, but could move around. The exact location of each part was treated as a latent variable, that was estimated as part of the object detection problem, using EM.

Generic object detection with deformable part-based models



Many slides based on P. Felzenszwalb