Majorization-Minimization (MM) algorithms optimize an objective function by iteratively minimizing its majorizing surrogate and offer attractively fast convergence rate for convex problems. However, their convergence behaviors for non-convex problems remain unclear. we propose a novel MM surrogate function from strictly upper bounding the objective to bounding the objective in expectation.

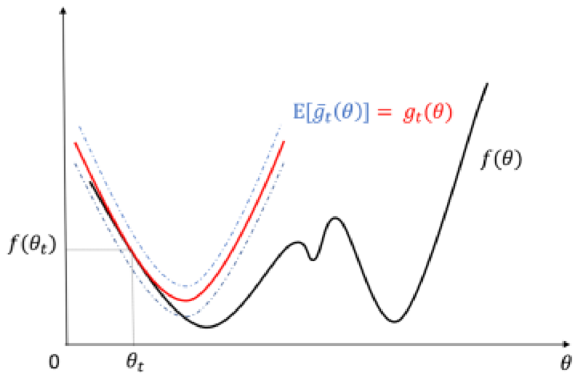
finite-sum optimization problem:

$$\min_{\boldsymbol{\theta}} \left[\Phi(\boldsymbol{\theta}) \triangleq \frac{1}{n} \sum_{i=1}^{n} f_i(\boldsymbol{\theta}) + h(\boldsymbol{\theta}) \right],$$

each component fi : R^p→ R is a continuous, nonconvex but smooth function, associated with one sample. The second term h(θ) could be non-smooth and non-convex. n is the number of samples.

MM algorithms solve problem via two steps. The first step is to construct a proper surrogate g that upper bounds the objective function Φ(θ) tightly, i.e., g(θ) ≥ Φ(θ). The second step is to optimize the surrogate whose optimum is much easier to obtain. Since the surrogate constructed at the current estimator is majorant to the objective function, each minimization step over the surrogate will decrease the objective function monotonically.

Here we use SPIDER (Stochastic Path Integrated Differential EstimatoR) method and develop a new MM algorithm, called SPI-MM. Instead of tightly bounding the objective for all the samples, it only requires the surrogate to bound the objective in expectation.



The red line is the classic surrogate which strictly upper bounds the objective function at the solution θt.

SPI-MM employs a specific generalized surrogate construction.

Our proposed SPI-MM algorithm fully leverages the gradient of past surrogates as follows.Suppose each epoch includes p iterations and totally we need T iterations. At the first step t0 in each epoch, we use all the samples n to construct a strict surrogate as in classic MM. For each datum, we choose a surrogate gi of Φi near θ0. When h(θ) in Φ(θ) is non-smooth, we choose a surrogate gi of fi near θ0.

We define |S1| = n. Then we have

$$\bar{g}_0(\boldsymbol{\theta};\boldsymbol{\theta}_0) = \frac{1}{|\mathcal{S}_1|}\sum_{i\in\mathcal{S}_1} g_i.$$

We minimize Eqn to get θ1. For the next step, we only sample S2(Mini batch size) samples. Relying on θ0, θ1 respectively, we construct a base surrogate gS2 (θ; θ1) of Φi ∈ S2 near θ1 and an auxiliary surrogate gS2 (θ; θ0) near θ0. These two surrogates here at least satisfy the this condition(A function

g : R^p → R is a first-order surrogate function of Φ in problem (1) near θt when: g(θt; θt) = Φ(θt); ∇g(θt; θt) = ∇Φ(θt) when Φ(θ) is smooth.)

We use the base surrogate gS2 (θ; θ1) as the first term of the new surrogate in this step. We compute the gradient of the auxiliary surrogate gS2 (θ; θ0) and ǵ0(θ; θ0) at θ0, i.e. ∇gS2 (θ0; θ0), ∇g¯0(θ0; θ0) to construct a linear term (−∇gS2 (θ0; θ0) + ∇g¯0(θ0; θ0))>(θ −θ1), and take it as the second term in the new surrogate. We add a second-order term $\frac{\mu}{2}||\theta 0 - \theta 1||^2$ to complement for the relaxing in the base surrogate. We minimize the sum of these three terms to get θ2. The following optimization iterations repeat the above surrogate construction, summarized as

$$\begin{aligned}\bar{g}_t(\boldsymbol{\theta};\boldsymbol{\theta}_t) = g_{\mathcal{S}_2^t}(\boldsymbol{\theta};\boldsymbol{\theta}_t) + \{&-\nabla g_{\mathcal{S}_2^t}(\boldsymbol{\theta}_{t-1};\boldsymbol{\theta}_{t-1}) \\ &+ \sum_{i=1}^{t-1}[\nabla g_{\mathcal{S}_2^i}(\boldsymbol{\theta}_i;\boldsymbol{\theta}_i) - \nabla g_{\mathcal{S}_2^i}(\boldsymbol{\theta}_{i-1};\boldsymbol{\theta}_{i-1})] \\ &+ \nabla\bar{g}_0(\boldsymbol{\theta}_0,\boldsymbol{\theta}_0)\}^\top(\boldsymbol{\theta}-\boldsymbol{\theta}_t) + \frac{\mu}{2}\|\boldsymbol{\theta}-\boldsymbol{\theta}_t\|^2.\end{aligned}$$

We rewrite the gradient part in the linear term as$-\nabla gS_t^2\ (\theta t - 1;\ \theta t - 1) +\ V_{t-1}$ for simplicity:

$$\mathcal{V}_{t-1} \triangleq \sum_{i=1}^{t-1}[\nabla g_{\mathcal{S}_2^i}(\boldsymbol{\theta}_i;\boldsymbol{\theta}_i) - \nabla g_{\mathcal{S}_2^i}(\boldsymbol{\theta}_{i-1};\boldsymbol{\theta}_{i-1})] + \nabla\bar{g}_0(\boldsymbol{\theta}_0,\boldsymbol{\theta}_0).$$

Samples {S_2^{t-1}, S_2^{t-2}, ..., S_2^1 } in different iterations are equal in number. The idea for constructing this surrogate is borrowed from SPIDER, where current gradient is estimated by utilizing the gradient in the past to reduce the variance, leading to smaller sampling numbers. We demonstrate below that the surrogate in SPI-MM satisfies conditions in our Definition(Generalized surrogate). For the first condition, we have

g¯t(θt; θt) =gS_2^t (θt; θt)

from above Eqn. On the other side : gS_2^t (θt; θt)= Φ(θt).

leading to g¯t(θt; θt) = Φ(θt) verifying the first condition(ggen(θt; θt) = Φ(θt))

For ∇g¯t(θ; θt) at θt, the gradient is computed a

∇g¯t(θt; θt)=∇ gS_2^t (θt; θt)−∇ gS_2^t (θt−1; θt−1)+V_{t-1}

It is easy to obtain the expectation of −∇ gS_2^t (θt−1; θt−1)+V_{t-1} is zero by iteratively unfolding this term and i.i.d. sampling condition. Thus, we have E∇g¯t(θt; θt) =∇ gS_2^t (θt; θt)= ∇Φ(θt), which satisfies the 2nd condition((Generalized surrogate): E∇ggen(θt; θt) = ∇Φ(θt), if Φ(θ) is smooth,.)

Finally, for the 3rd condition(Eggen(θ; θt) ≥ Φ(θ) for all θ) we have Eg¯t(θ, θt) = $\mathrm{EgS}_2^{\mathrm{t}}$ (θ−θt)+ $\frac{\mu}{2}\|\theta 0 - \theta 1\|^2$.

Suppose the objective function Φ(θ) is L-smooth, ∇^2 $\mathrm{EgS}_2^{\mathrm{t}}$ $(\theta - \theta t)$ ≥µf . Then we just need to tune µ large enough to make µ¯ = µ+µf larger than L to satisfy the 3rd condition(Eggen(θ; θt) ≥ Φ(θ) for all θ). We argue that µf here is reasonable, since the base surrogate $\mathrm{gS}_2^{\mathrm{t}}$ (θ; θt) is selected by ourselves. For simplicity, we can even directly select a strongly convex function here.

Based on the base surrogate, we use the gradient of past surrogates to construct a linear term, which is able to reduce the variance of the base surrogate. Such a composition also facilitates the convergence analysis.

(2

A *Gaussian Mixture* is a function that is comprised of several Gaussians, each identified by *k* ∈ {1,..., *K*}, where *K* is the number of clusters of our dataset. Each Gaussian *k* in the mixture is comprised of the following parameters:

- A mean μ that defines its centre.
- A covariance Σ that defines its width. This would be equivalent to the dimensions of an ellipsoid in a multivariate scenario.
- A mixing probability π that defines how big or small the Gaussian function will be.

we want to know what is the probability that a data point x_{n} comes from Gaussian *k*. We can express this as: p(z_{nk} = 1| x_{n})

z is a *latent variable* that takes only two possible values. It is one when **x** came from Gaussian *k*, and zero otherwise.

Let the parameters of our model be: θ={μ,π,Σ}

From the product rule of probabilities, we know that

$$p(\mathbf{x}_n, \mathbf{z}) = p(\mathbf{x}_n|\mathbf{z})p(\mathbf{z})$$

first we will need *p*(**x***n*)**,**with Marginalization :

$$p(\mathbf{x}_n) = \sum_{k=1}^{K} p(\mathbf{x}_n|\mathbf{z})p(\mathbf{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)$$

we need to use an iterative method to estimate the parameters.

Because to find the optimal parameters for the Gaussian mixture, all we have to do is to differentiate this equation with respect to the parameters and there is a logarithm in equation of p(x) that affected the summation which Calculating the derivative of this expression and then solving for the parameters is going to be very hard,then it's easier to optimize p(x,z).

$$p(\mathbf{x}_n) = \sum_{k=1}^{K} p(\mathbf{x}_n|\mathbf{z})p(\mathbf{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)$$

$$p(\mathbf{X}) = \prod_{n=1}^{N} p(\mathbf{x}_n) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)$$

$$\ln p(\mathbf{X}) = \sum_{n=1}^{N} \ln \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)$$

Initialise θ accordingly. For instance, we can use the results obtained by a previous K-Means run as a good starting point for our algorithm.

Then evaluate:

$$\mathcal{Q}(\theta^*, \theta) = \mathbb{E}[\ln p(\mathbf{X}, \mathbf{Z}|\theta^*)] = \sum_{\mathbf{Z}} p(\mathbf{Z}|\mathbf{X}, \theta) \ln p(\mathbf{X}, \mathbf{Z}|\theta^*)$$

We know

$$p(z_k = 1|\mathbf{x}_n) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n|\mu_j, \Sigma_j)} = \gamma(z_{nk})$$

Then we have:

$$\mathcal{Q}(\theta^*, \theta) = \sum_{\mathbf{Z}} \gamma(z_{nk}) \ln p(\mathbf{X}, \mathbf{Z}|\theta^*)$$

$p(\mathbf{X}, \mathbf{Z}|\theta^*)$ It is just the complete likelihood of the model, including both **X** and **Z**, and we can find it by using the following expression:

$$p(\mathbf{X}, \mathbf{Z}|\theta^*) = \prod_{n=1}^{N} \prod_{k=1}^{K} \pi^{z_{nk}} \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)^{z_{nk}}$$

Which is the result of calculating the joint probability of all observations and latent variables and is an extension of our initial derivations for *p*(**x**). The log of this expression is given by

$$\ln p(\mathbf{X}, \mathbf{Z}|\theta^*) = \sum_{n=1}^{N}\sum_{k=1}^{K} z_{nk}[\ln \pi_k + \ln \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)]$$

we have finally gotten rid of this troublesome logarithm that affected the summation . With all of this in place, it will be much easier for us to estimate the parameters by just maximizing *Q* with respect to the parameters, but we will deal with this in the *maximization step*. Besides, the latent variable *z* will **only** be 1 once everytime the summation is evaluated. With that knowledge, we can easily get rid of it as needed for our derivations.

Finally we have:

$$\mathcal{Q}(\theta^*, \theta) = \sum_{n=1}^{N}\sum_{k=1}^{K} \gamma(z_{nk})[\ln \pi_k + \ln \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)]$$

In the maximization step, we will find the revised parameters of the mixture.

$$\theta^* = \arg\max_\theta \mathcal{Q}(\theta^*, \theta)$$

Where

$$\mathcal{Q}(\theta^*, \theta) = \sum_{n=1}^{N}\sum_{k=1}^{K} \gamma(z_{nk})[\ln \pi_k + \ln \mathcal{N}(\mathbf{x}_n|\mu_k, \Sigma_k)]$$

we will need to add a suitable Lagrange multiplier

$$\mathcal{Q}(\theta^*, \theta) = \sum_{n=1}^{N}\sum_{k=1}^{K} \gamma(z_{nk})[\ln \pi_k + \ln \mathcal{N}(x_n|\mu_k, \Sigma_k)] - \lambda\left(\sum_{k=1}^{K} \pi_k - 1\right)$$

And now we can easily determine the parameters by using maximum likelihood. Let’s now take the derivative of *Q* with respect to π and set it equal to zero:

$$\frac{\partial \mathcal{Q}(\theta^*, \theta)}{\partial \pi_k} = \sum_{n=1}^{N} \frac{\gamma(z_{nk})}{\pi_k} - \lambda = 0$$

Then, by rearranging the terms and applying a summation over *k* to both sides of the equation, we obtain:

$$\sum_{n=1}^{N} \gamma(z_{nk}) = \pi_k \lambda \implies \sum_{k=1}^{K}\sum_{n=1}^{N} \gamma(z_{nk}) = \sum_{k=1}^{K} \pi_k \lambda$$

we know that the summation of all mixing coefficients π equals one. In addition, we know that summing up the probabilities γ over *k* will also give us 1. Thus we get *λ* = *N*. Using this result, we can solve for *π*:

$$\pi_k = \frac{\sum_{n=1}^{N} \gamma(z_{nk})}{N}$$

Similarly, if we differentiate *Q* with respect to μ and Σ, equate the derivative to zero and then solve for the parameters by making use of the log-likelihood equation (2) we defined, we obtain

$$\mu_k^* = \frac{\sum_{n=1}^{N} \gamma(z_{nk})\mathbf{x}_n}{\sum_{n=1}^{N} \gamma(z_{nk})}, \qquad \Sigma_k^* = \frac{\sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \mu_k)(\mathbf{x}_n - \mu_k)^T}{\sum_{n=1}^{N} \gamma(z_{nk})}$$

we will use these revised values to determine γ in the next EM iteration and so on and so forth until we see some convergence in the likelihood value.

(3

Variational Bayesian methods are a family of techniques for approximating intractable integrals arising in Bayesian inference and machine learning. They are typically used in complex statistical models consisting of observed variables (usually termed "data") as well as unknown parameters and latent variables, with various sorts of relationships among the three types of random variables, as might be described by a graphical model. As typical in Bayesian inference, the parameters and latent variables are grouped together as "unobserved variables". Variational Bayesian methods are primarily used for two purposes:

1. To provide an analytical approximation to the posterior probability of the unobserved variables, in order to do statistical inference over these variables.
2. To derive a lower bound for the marginal likelihood (sometimes called the *evidence*) of the observed data (i.e. the marginal probability of the data given the model, with marginalization performed over unobserved variables). This is typically used for performing model selection, the general idea being that a higher marginal likelihood for a given model indicates a better fit of the data by that model and hence a greater probability that the model in question was the one that generated the data.

Variational Bayes can be seen as an extension of the expectation-maximization (EM) algorithm from maximum a posteriori estimation (MAP estimation) of the single most probable value of each parameter to fully Bayesian estimation which computes (an approximation to) the entire posterior distribution of the parameters and latent variables. As in EM, it finds a set of optimal parameter values, and it has the same alternating structure as does EM, based on a set of interlocked (mutually dependent) equations that cannot be solved analytically.

Compared with EM:

Variational Bayes (VB) is often compared with expectation maximization (EM). The actual numerical procedure is quite similar, in that both are alternating iterative procedures that successively converge on optimum parameter values. The initial steps to derive the respective procedures are also vaguely similar, both starting out with formulas for probability densities and both involving significant amounts of mathematical manipulations.

However, there are a number of differences. Most important is *what* is being computed.

- EM computes point estimates of posterior distribution of those random variables that can be categorized as "parameters", but only estimates of the actual posterior distributions of the latent variables (at least in "soft EM", and often only when the latent variables are discrete). The point estimates computed are the modes of these parameters; no other information is available.
- VB, on the other hand, computes estimates of the actual posterior distribution of all variables, both parameters and latent variables. When point estimates need to be derived, generally the mean is used rather than the mode, as is normal in Bayesian inference. Concomitant with this, the parameters computed in VB do *not* have the same significance as those in EM. EM computes optimum values of the parameters of the Bayes network itself. VB computes optimum values of the parameters of the distributions used to approximate the parameters and latent variables of the Bayes network. For example, a typical Gaussian mixture model will have parameters for the mean and variance of each of the mixture components. EM would directly estimate optimum values for these parameters. VB, however, would first fit a distribution to these parameters — typically in the form of a prior distribution, e.g. a normal-scaled inverse gamma distribution — and would then compute values for the parameters of this prior distribution, i.e. essentially hyperparameters. In this case, VB would compute optimum estimates of the four parameters of the normal-scaled inverse gamma distribution that describes the joint distribution of the mean and variance of the component.

(4

(a

EM algorithm models the data as being generated by mixture of Gaussians. The EM algorithm estimates the parameters of (mean and covariance matrix) of each Gaussian. Each Gaussian defines a single cluster of data.

Expectation-Maximization algorithm consists of three steps. Initialization, E-step, M-step. First we randomly divide the dataset into K different clusters and we start with M-step to find weights, mean, and covariance matrix. We use those values and apply E-step. We keep looping through these two steps until some threshold condition has met.

Suppose we have a dataset with n rows and d attributes. Let K be the number of clusters. Let x_1, x_2, ..., x_n be d-dimensional vectors. Then, x_i = (x_i,1 , x_i,2 , ..., x_i,d), where each x_i,j is a real number.

Firstly, we initialize a weight vector with size K. Then, we create a 2D array containing the mean values with size k x d. Where each row contains the mean value for each Gaussian. Then, we create a 3D array to store the values for covariance matrix with size k x d x d. Where each dimension represents a d x d covariance matrix for each Gaussian. Now, we randomly assign data to each Gaussian with a 2D probability matrix of n x k. Where, n is the number of data we have.

In the M-Step, we use the probabilities that we initialized earlier to find weights, mean, and covariance matrix of each Gaussian using the following equations.

$$\sigma_{i,r,c} = \frac{\sum_{j=1}^{n}[p_{ij}(x_{j,r} - \mu_{i,r})(x_{j,c} - \mu_{i,c})]}{\sum_{j=1}^{n} p_{ij}}$$

$$\mu_i = \frac{\sum_{j=1}^{n}[p_{ij}\, x_j]}{\sum_{j=1}^{n} p_{ij}}$$

$$w_i = \frac{\sum_{j=1}^{n} p_{ij}}{\sum_{i=1}^{k}[\sum_{j=1}^{n} p_{ij}]}$$

b)In the E-step, we will use the weights, mean, and covariance matrix to adjust the values of probability using Gaussian estimation formula shown below.

$$N(v) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} \exp\left(-\frac{1}{2}(x - \mu)^{\mathrm{T}}\Sigma^{-1}(x - \mu)\right)$$

In the above equation, E is the covariance matrix, and u is the mean value. Now the new probability will be calculated as follows.

$$p_{ij} = \frac{N_i(x_j) * w_i}{P(x_j)}$$

Probability of data x in i Gaussian

$$P(x_j) = \sum_{i=1}^{k} N_i(x_j) * w_i$$

Lastly, we calculate the the log likelihood of the data after each iteration using the following equation and we keep repeating M-step and E-step until last_log_likelihood = log_likelihood.

$$L(x_1, \dots, x_n) = \sum_{j=1}^{n} \log_2(M(x_j))$$

Log likelihood of data:

$$p(\mathrm{X} \mid \theta) = \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp - \frac{(x_n - \mu_k)^2}{2\sigma_k^2},$$

$$\mathrm{L} = \log p(\mathrm{X} \mid \theta) = \sum_{n=1}^{N} \log\left[\sum_{k=1}^{K} \pi_k \frac{1}{\sqrt{2\pi\sigma^2}} \exp - \frac{(x_n - \mu_k)^2}{2\sigma_k^2}\right],$$

$$\frac{\partial \mathrm{L}}{\partial \mu_k} = \sum_{n=1}^{N} \frac{\pi_k \mathrm{N}(x_n \mid \mu_k, \sigma_k)}{\sum_{j=1}^{K} \pi_j \mathrm{N}(x_n \mid \mu_j, \sigma_j)} \cdot \frac{x_n - \mu_k}{\sigma_k^2}.$$

(c

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\mu_j}} = \sum_{i=1}^{n} \frac{\partial \log p(\boldsymbol{x_i} | \boldsymbol{\theta})}{\partial \boldsymbol{\mu_j}} = \mathbf{0}^T \qquad (7)$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Sigma_j}} = \sum_{i=1}^{n} \frac{\partial \log p(\boldsymbol{x_i} | \boldsymbol{\theta})}{\partial \boldsymbol{\Sigma_j}} = \mathbf{0} \qquad (8)$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\pi_j}} = \sum_{i=1}^{n} \frac{\partial \log p(\boldsymbol{x_i} | \boldsymbol{\theta})}{\partial \boldsymbol{\pi_j}} = 0 \qquad (9)$$

Set of equations defined to optimize log-likelihood of the data over θ

Observe that the computation of each parameter from $\boldsymbol{\theta}$ ($\boldsymbol{\mu}$'s, $\boldsymbol{\Sigma}$'s, $\boldsymbol{\pi}$'s) depends on the other parameters in a complex way. Consequently, there is no closed-form solution to these equations.

To use the strategy of optimizing some parameters while keeping the others fixed,we use Expectation-Maximization algorithm.

Responsibility

the responsibility $\boldsymbol{r_{ij}}$ is the probability that the $\boldsymbol{j}$th mixture component generated point $\boldsymbol{x_i}$.

$$r_{ij} := \frac{\pi_j \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_j}, \boldsymbol{\Sigma_j})}{\sum_{j'=1}^{K} \pi_{j'} \mathcal{N}(\boldsymbol{x_i}|\boldsymbol{\mu_{j'}}, \boldsymbol{\Sigma_{j'}})}$$

we can define the total responsibility of the *j*th mixture component for the entire dataset:

$$N_j := \sum_{i=1}^{n} r_{ij}$$

there's a consistent way of updating the individual parameters of a GMM given prior (initialized at random) parameters **μ**'s, **Σ**'s, and **π**'s. This approach works by updating some parameters while keeping the others fixed. Below Equations summarizes the equations used for that purpose:

$$\hat{\boldsymbol{\mu}}_j = \frac{1}{N_j} \sum_{i=1}^{n} r_{ij} \boldsymbol{x}_i$$

$$\hat{\boldsymbol{\Sigma}}_j = \frac{1}{N_j} \sum_{i=1}^{n} r_{ij} (\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_j)(\boldsymbol{x}_i - \hat{\boldsymbol{\mu}}_j)^T$$

$$\hat{\pi}_j = \frac{N_j}{N}, \quad j = 1, \cdots, K$$

Update of the parameters of the GMM

the update of μ, Σ, and π, all depend on the responsibilities ($\boldsymbol{r}_{ij}$), which by its turn, depends on μ, Σ, and π. That's why there's not a closed-form solution to equations.

(5

we say that a distribution f is a mixture of K component distributions f1, f2,... fK if

$$f(x) = \sum_{k=1}^{K} \lambda_k f_k(x)$$

In parametric mixture models, where the fk are all from the same parametric family, but with different parameters — for instance they might all be Gaussians with different centers and variances, or all Poisson distributions with different means, or all power laws with different exponents. (It's not strictly necessary that they all be of the same kind.) We'll write the parameter, or parameter vector, of the kth component as θk , so the model becomes:

$$f(x) = \sum_{k=1}^{K} \lambda_k f(x;\theta_k)$$

The over-all parameter vector of the mixture model is thus θ = (λ1,λ2,...λK,θ1,θ2,...θK).

Let's consider two extremes. When K = 1, we have a simple parametric distribution, of the usual sort, and density estimation reduces to estimating the parameters, by maximum likelihood or whatever else we feel like. On the other hand when K = n, the number of observations, we have gone back towards kernel density estimation. If K is fixed as n grows, we still have a parametric model, and avoid the curse of dimensionality, but a mixture of (say) ten Gaussians is more flexible than a single Gaussian — thought it may still be the case that the true distribution just can't be written as a ten-Gaussian mixture. So we have our usual bias-variance or accuracy precision trade-off — using many components in the mixture lets us fit many distributions very accurately, with low approximation error or bias, but means we have more parameters and so we can't fit any one of them as precisely, and there's more variance in our estimates.

it's generally a good idea to estimate distributions using maximum likelihood, when we can. How could we do that here? Remember that the likelihood is the probability (or probability density) of observing our data, as a function of the parameters. Assuming independent samples, that would be

$$\prod_{i=1}^{n} f(x_i;\theta)$$

or observations x1, x2,... xn. As always, we'll use the logarithm to turn multiplication into addition:

$$\begin{aligned} \ell(\theta) &= \sum_{i=1}^{n} \log f(x_i;\theta) \\ &= \sum_{i=1}^{n} \log \sum_{k=1}^{K} \lambda_k f(x_i;\theta_k) \end{aligned}$$

Let's try taking the derivative of this with respect to one parameter, say θj

$$\begin{aligned} \frac{\partial \ell}{\partial \theta_j} &= \sum_{i=1}^{n} \frac{1}{\sum_{k=1}^{K} \lambda_k f(x_i;\theta_k)} \lambda_j \frac{\partial f(x_i;\theta_j)}{\partial \theta_j} \\ &= \sum_{i=1}^{n} \frac{\lambda_j f(x_i;\theta_j)}{\sum_{k=1}^{K} \lambda_k f(x_i;\theta_k)} \frac{1}{f(x_i;\theta_j)} \frac{\partial f(x_i;\theta_j)}{\partial \theta_j} \\ &= \sum_{i=1}^{n} \frac{\lambda_j f(x_i;\theta_j)}{\sum_{k=1}^{K} \lambda_k f(x_i;\theta_k)} \frac{\partial \log f(x_i;\theta_j)}{\partial \theta_j} \end{aligned}$$

If we just had an ordinary parametric model, on the other hand, the derivative of the log-likelihood would be

$$\sum_{i=1}^{n} \frac{\partial \log f(x_i;\theta_j)}{\partial \theta_j}$$

So maximizing the likelihood for a mixture model is like doing a weighted likelihood maximization, where the weight of xi depends on cluster, being

$$w_{ij} = \frac{\lambda_j f(x_i;\theta_j)}{\sum_{k=1}^{K} \lambda_k f(x_i;\theta_k)}$$

The problem is that these weights depend on the parameters we are trying to estimate.

Let’s look at these weights wi j a bit more. Remember that λj is the probability that the hidden class variable Z is j , so the numerator in the weights is the joint probability of getting Z = j and X = xi . The denominator is the marginal probability of getting X = xi , so the ratio is the conditional probability of Z = j given X = xi ,

$$w_{ij} = \frac{\lambda_j f(x_i;\theta_j)}{\sum_{k=1}^{K} \lambda_k f(x_i;\theta_k)} = p(Z = j|X = x_i;\theta)$$

If we try to estimate the mixture model, then, we’re doing weighted maximum likelihood, with weights given by the posterior cluster probabilities. These, to repeat, depend on the parameters we are trying to estimate, so there seems to be a vicious circle.

A crude way of doing this would start with an initial guess about the component distributions; find out which component each point is most likely to have come from; re-estimate the components using only the points assigned to it, etc., until things converge. This corresponds to taking all the weights wi j to be either 0 or 1. However, it does not maximize the likelihood, since we’ve seen that to do so we need fractional weights. What’s called the EM algorithm is simply the obvious refinement of this “hard” assignment strategy.

1. Start with guesses about the mixture components θ1,θ2,...θK and the mixing weights λ1,...λK.

2. Until nothing changes very much:

(a) Using the current parameter guesses, calculate the weights wi j (E-step)

(b) Using the current weights, maximize the weighted likelihood to get new parameter estimates (M-step)

3. Return the final parameter estimates (including mixing proportions) and cluster probabilities

The M in “M-step” and “EM” stands for “maximization”, which is pretty transparent. The E stands for “expectation”, because it gives us the conditional probabilities of different values of Z, and probabilities are expectations of indicator functions. (In fact in some early applications, Z was binary, so one really was computing the expectation of Z.) The whole thing is also called the “expectation-maximization” algorithm.

FATEME SALEHIN 97101961