ECSE 343: Numerical Methods in Engineering

Assignment 2: Advanced Model Fitting

Due: Thursday, February 17th, 2022 at 11:59pm EST on myCourses
Final weight: **15%**

Please follow the *same* policies and processes as the previous assignments (refer to section 1 in the A0 handout).

The learning objectives of this assignment are to explore and implement a:

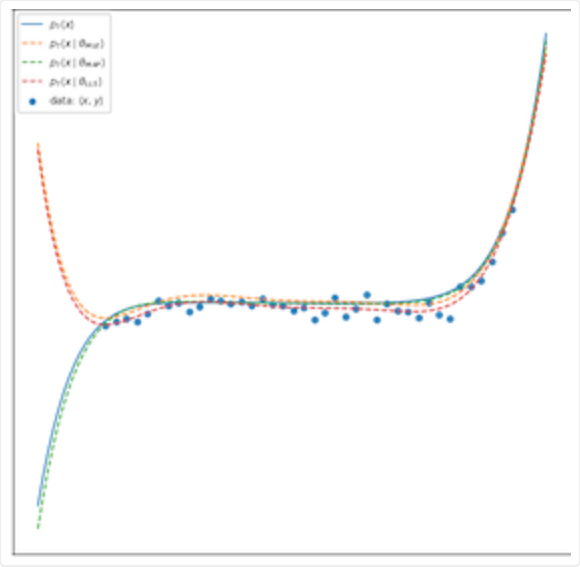
- maximum likelihood estimator,
- maximum a posteriori estimator, and
- the expectation-maximization algorithm.

Contents

In doing so, you'll revisit the polynomial fitting problem before moving on to a more sophisticated mixture model classification problem.

1 Polynomial regression with MLE and MAP [40%]

We begin by exploring a *maximum likelihood estimator* (MLE) and *maximum a posteriori* (MAP) estimator on the familiar *polynomial regression* problem.



Comparing degree-7 polynomial fits to noisy data and across estimators.

1.1 Polynomial regression

The overdetermined degree-m polynomial regression problem — with an explicit corruption/noise model on the data — seeks an interpolant across n data samples (x_i, y_i) that satisfies:

$$y_i = \sum_{j=0}^{m} \theta_j x_i^j + \epsilon_i \quad (i = 1, \dots, n)$$

where:

- x_i is the *independent coordinate* of sample i, with $\mathbf{x} = \{x_0 \dots x_{n-1}\}$,
- y_i is the *dependent coordinate* of sample i, with $\mathbf{y} = \{y_0 \dots y_{n-1}\}$,
- $\epsilon_i \sim \mathcal{N}(\mu, \sigma^2)$ is standard *Gaussian noise* corrupting the outputs y_i, and
- $\boldsymbol{\theta} = \{\theta_0, \theta_1, \dots, \theta_m\}$ are the $p = m + 1$ polynomial coefficients we are solving for.

Note that one common way to rewrite this model is by "folding in" the deterministic component into the mean of the Gaussian, as:

$$y_i \sim \mathcal{N}\left(\mu + \sum_{j=0}^{m} \theta_j x_i^j, \sigma^2\right) .$$

1.2 Maximum likelihood estimation (MLE) [20%]

You will implement a MLE for the polynomial parameters $\boldsymbol{\theta}$ that maximize the data's likelihood function:

$$\boldsymbol{\theta}_{\text{MLE}} := \underset{\boldsymbol{\theta}}{\text{argmax}}\ p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) ,$$

where — assuming the samples (x_i, y_i) are drawn i.i.d. — the likelihood function can be expressed using the normal distribution's density, as

$$p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta}) = \prod_{i=0}^{n-1} p(y_i|x_i, \boldsymbol{\theta}) = \prod_{i=0}^{n-1} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}\left[y_i - \mu - \sum_{j=0}^{m} \theta_j x_i^j\right]^2\right).$$

Taking the log of the likelihood before taking the argmax — which is a valid transformation under argmax, given log's monotonicity — yields:

$$\log(p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})) = \log\left(\prod_{i=0}^{n-1} p(y_i|x_i, \boldsymbol{\theta})\right) = -n\log(\sqrt{2\pi\sigma^2})) - \frac{1}{2\sigma^2}\sum_{i=0}^{n-1}\left[y_i - \mu - \sum_{j=0}^{m} \theta_j x_i^j\right]^2$$

which we optimize for by setting the derivatives w.r.t. the parameters to zero,

$$\frac{\partial}{\partial\theta_k}\log(p(\mathbf{y}|\mathbf{x}, \boldsymbol{\theta})) = \frac{1}{\sigma^2}\sum_{i=0}^{n-1} x_i^k \left[y_i - \mu - \sum_{j=0}^{m} \theta_j x_i^j\right] = 0 ,$$

before combining these partial derivatives to form a linear system $\mathbf{A}\,\boldsymbol{\theta}_{\text{MLE}} = \mathbf{b}$, with

$$A_{i,j} = \sum_{s=0}^{n-1} x_s^{i+j} \text{ and } b_i = \sum_{s=0}^{n-1} x_s^i (y_s - \mu) \; ; \; i,j \in [0, M]$$

Solving this system yields the MLE for $\boldsymbol{\theta}_{\text{MLE}}$.

Deliverable 1 [20%]
Complete the implementation of the function `PolRegMLE` that takes inputs p (equal to $m+1$), μ, $\mathbf{x}$, and $\mathbf{y}$ and outputs the system matrix $\mathbf{A}$ and RHS $\mathbf{b}$ for the MLE. Notice how the system does *not* depend on σ, as discussed in class.

Unlike the first part of A1, deliverables 1, 2 and 3 below won't *solve* the linear system, but only form it. You can explore `numpy`'s many linear system solving routines (e.g, `numpy.linalg.solve`) in order to compute and assess the solutions $\boldsymbol{\theta}_*$ of your various estimator systems.

1.3 Maximum a posteriori (MAP) [20%]

Maximum a posteriori (MAP) estimators consider an additional prior over the parameters and solve for parameters $\boldsymbol{\theta}_{\text{MAP}}$ that maximize the (log) posterior distribution of the parameters, given the data:

$$\boldsymbol{\theta}_{\text{MAP}} := \underset{\boldsymbol{\theta}}{\text{argmax}} \; p(\boldsymbol{\theta}|\mathbf{y},\mathbf{x}) = \underset{\boldsymbol{\theta}}{\text{argmax}} \; p(\mathbf{y}|\mathbf{x},\boldsymbol{\theta})\, p(\boldsymbol{\theta}) \; .$$

In the polynomial regression setting, we will assume that the $\boldsymbol{\theta}$ are drawn i.i.d from a normal distribution $\mathcal{N}(\mu_\theta, \sigma_\theta^2)$, so their joint prior probability density is given by:

$$p(\boldsymbol{\theta}) = \prod_{k=0}^{m} p(\theta_k) = \prod_{k=0}^{m} \frac{1}{\sqrt{2\pi\sigma_\theta^2}} \exp\left(-\frac{1}{2\sigma_\theta^2} (\theta_k - \mu_\theta)^2 \right) .$$

The derivative with respect to each parameter θ_k of $\log(p(\mathbf{y}|\mathbf{x},\boldsymbol{\theta})\, p(\boldsymbol{\theta})) \equiv l(\boldsymbol{\theta}|\mathbf{x},\mathbf{y})$ is

$$\frac{\partial}{\partial \theta_k} l(\boldsymbol{\theta}|\mathbf{x},\mathbf{y}) = \frac{1}{\sigma^2} \sum_{i=0}^{n-1} x_i^k \left[y_i - \mu - \sum_{j=0}^{m} \theta_j x_i^j \right] - \frac{1}{\sigma_\theta^2} (\theta_k - \mu_\theta) \; ,$$

and we combine these partials together to form a linear system $\mathbf{A}\boldsymbol{\theta}_{\text{MAP}} = \mathbf{b}$ that we can solve for the MAP estimate of the parameters, where

$$A_{i,j} = \begin{cases} \sum_{s=0}^{n-1} x_s^{i+j} + \dfrac{\sigma^2}{\sigma_\theta^2} & \text{if } i = j \\ \sum_{s=0}^{n-1} x_s^{i+j} & \text{otherwise} \end{cases} \quad \text{and } b_i = \sum_{s=0}^{n-1} x_s^i (y_s - \mu) + \mu_\theta \frac{\sigma^2}{\sigma_\theta^2} \; ; \; i,j \in [0, m]$$

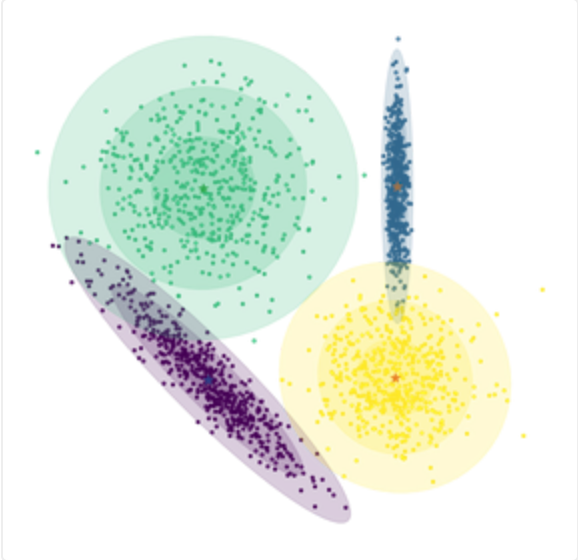
Deliverable 2 [15%]
Complete the implementation of the function `PolRegMAP` with inputs $p = m+1$, μ, σ, μ_θ, σ_θ, $\mathbf{x}$, and $\mathbf{y}$ and that outputs the MAP linear system matrix $\mathbf{A}$ and RHS $\mathbf{b}$.

As discussed in class, the linear least squares solution to the polynomial regression problem is equivalent to the MLE when the noise is assumed to be normally distributed with zero mean. As such, we should expect that MLE and LLS yield the same parameters — with the exception of the constant term θ_0 — when the noise is not of zero mean.

☑ **Deliverable 3 [5%]**
Revisit LLS by completing the implementation of `PolRegLLS` with inputs $p = m + 1$, n, $\mathbf{x}$ and $\mathbf{y}$ and that outputs the appropriate fully-constrained linear system matrix $\mathbf{A}$ and RHS $\mathbf{b}$ needed to construct the LLS system.

2 Mixture Models with Expectation-Maximization [60%]

We will now explore the *expectation-maximization* (EM) algorithm through the lens of a *Gaussian mixture model* (GMM) example.



Fitting a GMM to a randomly generated, multi-modal 2D dataset using the EM algorithm.

ⓘ Our evaluation of your GMM code will impose a (sufficiently lenient) time limit on the execution your algorithm — be mindful to avoid loops and to leverage `numpy` vectorization whenever suitable.

Consider many d-dimensional data samples $\mathbf{x}_i$ drawn from a *mixture model* parameterized by $\boldsymbol{\theta}$.

Our example will use a *mixture of multivariate normal distributions* $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ (i.e., high-dimensional, anisotropic Gaussians), with K *mixture components*, and so our $\boldsymbol{\theta} = \{\boldsymbol{\mu}_0, \ldots, \boldsymbol{\mu}_{K-1}, \boldsymbol{\Sigma}_0, \ldots, \boldsymbol{\Sigma}_{K-1}, \pi_0, \ldots, \pi_{K-1}\}$, where

- each of the K d-dimensional Gaussians are parameterized by a d-dimensional mean vector $\boldsymbol{\mu}_i$, a $d \times d$ covariance matrix $\boldsymbol{\Sigma}_i$ (see below), and
- the normalized *mixture proportions* π_c weight the amount of each of the K components in the mixture, with $\sum_{c=0}^{K-1} \pi_c = 1$.

We use the standard definition of a d-dimensional multivariate normal distribution, with density:

$$\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\intercal \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right],$$

and so the marginal distribution of $\mathbf{x}_i$ is:

$$p(\mathbf{x}_i|\boldsymbol{\theta}) = \sum_{c=0}^{K-1} \pi_c \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \,.$$

As such, the joint probability of n i.i.d. data samples $\widehat{\mathbf{x}} = \{\mathbf{x}_0, \ldots, \mathbf{x}_{n-1}\}$ is

$$p(\widehat{\mathbf{x}}|\boldsymbol{\theta}) = \prod_{i=0}^{n-1} \sum_{c=0}^{K-1} \pi_c \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \,.$$

And so, given the samples $\widehat{\mathbf{x}}$ and knowing only the number of mixture components K, our goal is to estimate the mixture's parameters $\boldsymbol{\theta}$ that best match the data: the means $\boldsymbol{\mu}_c$ (i.e., centers), covariances $\boldsymbol{\Sigma}_c$ (i.e., anisotropic ``shapes''), and weights π_c of the mixture components.

2.1 From MLE to Expectation-Maximization

A natural first step is to derive an MLE for $\boldsymbol{\theta}$. The likelihood function is simply the joint probability $p(\widehat{\mathbf{x}}|\boldsymbol{\theta})$ above, and so the log-likelihood is

$$\log p(\widehat{\mathbf{x}}|\boldsymbol{\theta}) = \sum_{i=0}^{n-1} \log\left(\sum_{c=0}^{K-1} \pi_c \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \right) .$$

We immediately run into difficulties when differentiating with respect to, say, $\boldsymbol{\mu}_k$:

$$\frac{\partial}{\partial \boldsymbol{\mu}_j} \log p(\widehat{\mathbf{x}}|\boldsymbol{\theta}) = \sum_{i=0}^{n-1} \left(\frac{\pi_j \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}{\sum_{c=0}^{K-1} \pi_c \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)} \right) (\mathbf{x}_i - \boldsymbol{\mu}_j) \boldsymbol{\Sigma}_j^{-1} = 0 \,,$$

as we are unable to isolate the $\boldsymbol{\mu}_k$: the non linearity induced by the $\left(\pi_j \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\right) \Big/ \left(\sum_{c=0}^{K-1} \pi_c \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \right)$ term. We face similar problems when differentiating with respect to other parameters in $\boldsymbol{\theta}$.

To sidestep this issue, we will introduce the concept of a *latent label* $z_i \in \{1, \ldots, K\}$ for each data sample $\mathbf{x}_i$: this latent label corresponds to the ground truth association of each data sample to its corresponding mixture component, and so $\pi_c = p(z_i = c|\boldsymbol{\theta})$.

By the *law of total probability*, the marginal conditional probability distribution of $\mathbf{x}_i$ given $\boldsymbol{\theta}$ can now be re-written as

$$p(\mathbf{x}_i|\boldsymbol{\theta}) = \sum_{c=0}^{K-1} \underbrace{p(z_i = c|\boldsymbol{\theta})}_{\pi_c} \underbrace{p(\mathbf{x}_i|z_i = c, \boldsymbol{\theta})}_{\mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)} .$$

Intuitively, knowing the latent variables z_i should help us in computing the MLE. To do so, we first compute the vector posterior $\boldsymbol{\alpha}$ with elements $\alpha_{i,j} = p(z_i = k|\mathbf{x}_i, \boldsymbol{\theta})$ using Bayes' theorem:

$$\alpha_{i,j} := p(z_i = j|\mathbf{x}_i, \theta) = \frac{p(z_i = j|\theta)p(\mathbf{x}_i|z_i = j, \theta)}{p(\mathbf{x}_i|\theta)} = \frac{\pi_j \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}{\sum_{c=0}^{K-1} \pi_c \mathcal{N}(\mathbf{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)} .$$

We can now rewrite the partial derivative of the log-likelihood with respect to $\boldsymbol{\mu}_j$ and set it to zero as:

$$\frac{\partial}{\partial \boldsymbol{\mu}_j} \log p(\hat{\mathbf{x}}|\theta) = \sum_{i=0}^{n-1} \alpha_{i,j}(\mathbf{x}_i - \boldsymbol{\mu}_j)\boldsymbol{\Sigma}_j^{-1} = 0 .$$

Even though $\alpha_{i,j}$ depends on $\boldsymbol{\mu}_j$, we can define an **iterative process** that *assumes* we *have* an approximate value of $\alpha_{i,j}$ and solving for $\boldsymbol{\mu}_j$ as:

$$\boldsymbol{\mu}_j = \frac{1}{\omega_j} \sum_{i=0}^{n-1} \alpha_{i,j} \mathbf{x}_i$$

where $\omega_j = \sum_{i=0}^{n-1} \alpha_{i,j}$ can be interpreted as the effective number of data samples assigned to the mixture component j. We can similarly obtain expressions for the other parameters as:

$$\boldsymbol{\Sigma}_j = \frac{1}{\omega_j} \sum_{i=0}^{n-1} \alpha_{i,j}(\mathbf{x}_i - \boldsymbol{\mu}_j)^\intercal(\mathbf{x}_i - \boldsymbol{\mu}_j) \quad \text{and} \quad \pi_j = \frac{\omega_j}{n} .$$

After computing these (updated) values for the $\boldsymbol{\mu}_j$, $\boldsymbol{\Sigma}_j$ and π_j, we can recompute new estimates for $\alpha_{i,j}$ from the expression we derived using Bayes' theorem. We repeat this process until "convergence".

2.2 Formalizing the EM Algorithm

To summarize:

- if we knew the parameters $\boldsymbol{\theta}$, we could compute the posterior probabilities $\boldsymbol{\alpha}$, and
- if we knew the posteriors $\boldsymbol{\alpha}$, we could compute the parameters $\boldsymbol{\theta}$.

The EM algorithm proceeds as follows:

0. **Initialize** the parameters $\boldsymbol{\theta}$ and evaluate the log-likelihood $\log p(\hat{\mathbf{x}}|\boldsymbol{\theta})$ with these parameters,
1. **E-step**: evaluate the posterior probabilities $\alpha_{i,j}$ using the current values of the parameters $\boldsymbol{\theta}$,
2. **M-step**: estimate new parameters $\boldsymbol{\theta}$ with the current values of $\alpha_{i,j}$,
3. **Evaluate** the log-likelihood $\log p(\hat{\mathbf{x}}|\boldsymbol{\theta})$ with the new parameter estimates — if the log-likelihood has changed by less than some (typically small) convergence threshold, terminate; otherwise, repeat steps 1 through 3.

EM is sensitive to the initial parameter values, so special care must be taken in the first step. Given "valid" initial values, however, the EM algorithm guarantees that the log-likelihood increases after every iteration.

During EM iterations with GMMs, covariance matrices may become singular. To circumvent this problem, consider regularizing the covariance matrices, when necessary; note, however, that using too large a regularization can lead to divergence (e.g., a possible **reduction** in the log-likelihood estimate between iterations.)

We provide a basic class structure to encapsulate GMM parameters and some utility methods.

When completing your implementation, you should only require explicit loops *over the mixture component*. All other operations can be vectorized with `numpy` routines, such as `sum( , axis=)`, `dot( , axis=)`, `argmax( , axis=)` and the component-wise array operators `+`, `-` , `*` and `/`. We similarly provide you with access to `scipy`'s `multivariate_normal` class whose (efficient and vectorized) `pdf` method implements the Gaussian density evaluation.

Carefully review the `GMM` class we provide in the base code.

2.3 EM Implementation

Compute normalization of the mixture for all the samples

Compute the mixture normalization factors $\boldsymbol{\beta} = \{\beta_0, \ldots, \beta_{n-1}\}$, given the current parameters $\boldsymbol{\theta}$, for all the data samples, as:

$$\beta_i := \sum_{c=0}^{K-1} \pi_c \mathcal{N}(\mathbf{x}_i \,|\, \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \,.$$

You will use the result of this function in the expectation step and log-likelihood computation.

Deliverable 4 [10%]
Complete the implementation of `normalization` which takes as input the GMM instance and updates its $\boldsymbol{\beta}$ property `gmm.beta`.

Expectation step

Using the β_i values of $\boldsymbol{\beta}$, you have to compute the posterior probabilities $\boldsymbol{\alpha}$ with elements $\alpha_{i,j}$ for the mixture components, as:

$$\alpha_{i,j} = \frac{\pi_j \mathcal{N}(\mathbf{x}_i \,|\, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}{\beta_i}$$

Deliverable 5 [10%]
Complete the implementation of `expectation` which takes as input the GMM instance and updates its $\boldsymbol{\alpha}$ property `gmm.alpha`.

Maximization step

Using the $\alpha_{i,j}$ values of $\boldsymbol{\alpha}$ you have to update the following quantities:

$$z_i = \underset{j}{\operatorname{argmax}}\, \alpha_{i,j} \qquad \omega_j = \sum_{i=0}^{n-1} \alpha_{i,j} \qquad \pi_j = \frac{\omega_j}{n} \qquad \boldsymbol{\mu}_j = \frac{1}{\omega_j} \sum_{i=0}^{n-1} \alpha_{i,j}\, \mathbf{x}_i$$

$$\boldsymbol{\Sigma}_j = \frac{1}{\omega_j} \sum_{i=0}^{n-1} \alpha_{i,j} (\mathbf{x}_i - \boldsymbol{\mu}_j)^{\mathsf{T}} (\mathbf{x}_i - \boldsymbol{\mu}_j) + \boldsymbol{\Sigma}_{\text{regularization}}$$

Deliverable 6 [20%]
Complete the implementation of `maximization` which updates the z_i, ω_j, π_j, $\boldsymbol{\mu}_j$, $\boldsymbol{\Sigma}_j$ in the input GMM instance; the associated member variables are `gmm.Z`, `gmm.weight`, `gmm.pi`, `gmm.mu`, `gmm.cov`. Afterwards, you can update the multivariate normal parameters of the mixture elements in `gmm.gauss` with $\boldsymbol{\mu}_j$ and $\boldsymbol{\Sigma}_j$.

Don't forget to regularize the covariance matrices.

Computing the log-likelihood

We can finally compute the log-likelihood estimate using the current $\boldsymbol{\beta}$:

$$\log P(\hat{\mathbf{x}}|\boldsymbol{\theta}) = \sum_{i=0}^{n-1} \log(\beta_i)$$

Deliverable 7 [10%]
Complete the implementation of `logLikelihood` which computes the log-likelihood $\log P(\hat{\mathbf{x}}|\boldsymbol{\theta})$ returning nothing and updating the GMM instance variable `gmm.log_likelihoods`.