

Expectation-Maximization (EM)

Saturday, October 6, 2018 12:21

For the glory of God

Maximum Likelihood Estimation (MLE)

- As we know, Maximum Likelihood Estimation (MLE) attempts to find the parameter value that maximizes the **likelihood**.

Given observed data, what is the chance that a given reality is true? ↓


- Basically, what MLE does is to maximize **log-likelihood** function to estimate the parameter value.

↓
we need to take logarithm because maximization of the likelihood function is difficult in general.

- If we assume the distribution as Gaussian distribution, the log-likelihood function is as following;

a) For univariate Gaussian distribution 

$$P(X|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2} \frac{(X-\mu)^2}{\sigma^2}\right) ; \text{ where } \sigma^2 = \text{Variance}, \mu = \text{mean}$$

b) For multi-variate Gaussian distribution 

$$P(X|\mu, \Sigma) = \frac{1}{((2\pi)^{D/2} |\Sigma|)^{1/2}} \exp\left[-\frac{1}{2} (X-\mu)^T \Sigma^{-1} (X-\mu)\right] ; \text{ where } \Sigma = \text{covariance}, D = \text{Dimension}$$

- In the end, MLE will provide the best parameters (μ, σ^2, Σ) that fit well given data (X) .
- Then, how does MLE find the best parameters?
 - The basic idea is to take the derivative and to equate it to zero.

$$\frac{\partial P(X|\theta)}{\partial \theta} = 0 \rightarrow \theta_{MLE} ; \text{ where } \theta \text{ represents parameters } (\mu, \sigma^2, \dots)$$

$$\Leftrightarrow \theta_{MLE} = \underset{\theta}{\operatorname{argmax}} P(X|\theta)$$

- For example, let's say that we have N number of data points. (x_1, x_2, \dots, x_N)

$$\theta_{MLE} = \underset{\theta}{\operatorname{argmax}} P(x_i|\theta)$$

$$= \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^N P(x_i|\theta) ; \text{ Here, we assumed the data points are IID (Independently Identically Distributed)}$$

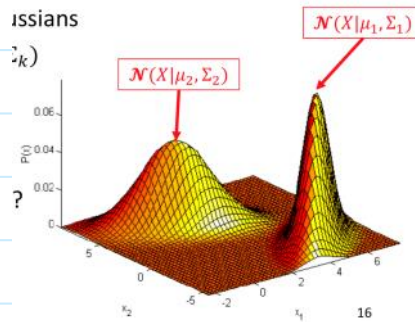
$$= \underset{\theta}{\operatorname{argmax}} \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right)^N \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^N (x_i - \mu)^2\right) ; \text{ if } p \text{ is univariate Gaussian distribution with the data points}$$

Gaussian Mixture Model (GMM)

- The probability (p) may be multi-modal as a mixture of uni-modal distributions.

$$p(x) = \sum_{k=1}^K \pi_k \mathcal{N}(x | \mu_k, \Sigma_k) \quad ; \text{ where } K \text{ is number of Gaussians}$$

mixing coefficient : weightage for each Gaussian distribution ; $0 \leq \pi_k \leq 1$ and $\sum_{k=1}^K \pi_k = 1$



: two Gaussian distributions

e.g. $\pi_1 = 0.3$ / $\pi_2 = 0.7$

- Let's say that we want to maximize likelihood of the multi-modal distribution.
- In a similar way, we would take logarithm and derivative ;

$$\ln p(x) = \sum_{n=1}^N \ln \left(\sum_{k=1}^K \pi_k \mathcal{N}(x_n | \mu_k, \Sigma_k) \right)$$

$$\frac{d}{d\theta} \ln p(x|\theta) = ? \Rightarrow \text{MLE doesn't work because there is no closed form solution.}$$

(Non-convex difficult)

- Therefore, for this case, parameters can be calculated using EM algorithm.

Expectation - Maximization (EM) Algorithm based on GMM

Introduction

- The EM algorithm was explained and given its name in 1977.
- The EM algorithm has been used to find MLE of parameters in a statistical model (e.g. Gaussian) where the equation can't be solved directly.
- Typically, the model involves latent variables. (Hidden variable)
- Thus, I would say that ;
 - The EM algorithm is an efficient iterative procedure to compute the MLE in the presence of hidden data.

Latent variable

- Statistically, latent variables are variables that cannot be observed directly but its values can be inferred by taking other measurements.
- e.g. we may not be able to directly quantify intelligence but we think it exists.

↪ Your intelligence is a latent variable that affects your performance on multiple tasks even if it can't be directly measured.

- When we think about the probability,

- We can think of the mixing coefficients as prior probabilities. $p(k) = \pi_k$

- For a given value of x , we can evaluate the corresponding posterior probabilities. $r_k(x) = p(k|x)$

⇒ This is called as responsibilities $r_k(x)$. ; Then r = latent variable

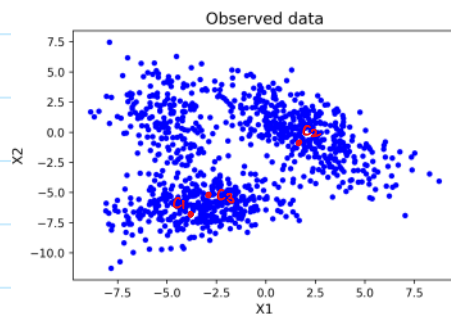
$$p(k|x) = \frac{p(k)p(x|k)}{p(x)} \quad ; \text{Based on Bayes rule}$$

$$= \frac{\pi_k N(x|\mu_k, \Sigma_k)}{\sum_{i=1}^K \pi_i N(x|\mu_i, \Sigma_i)} \quad ; \text{where } \pi_k = \frac{N_k}{N} \quad ; N_k = \text{number of data points assigned to cluster } k.$$

How EM algorithm works

Step 1) Initialization

- Let's assume that $K=3$.
- As the first step, we will place three points randomly on the observed data.



Python code

```
[N,d] = X.shape
K = 3
C = np.zeros([K,d])
for i in range(K):
    random_index = np.random.randint(N)
    C[i] = X[random_index]
```

The points are on the observed data.

; where X = data set, random index would give you K random points

- As the second step, we will initialize the values of μ_i , Σ_i , and π_i .

; For mixing coefficient (π_i), $\pi_i = 1/K \quad \therefore \pi_1 = 0.3, \pi_2 = 0.3, \pi_3 = 0.3$

; For mean (μ_i), $\mu_i = C[i]$; it means the centroids are considered as a mean.

$$\text{e.g. } C_1 = \mu_1 = [-3.16, -6.45]$$

; For covariance (Σ_i), $\Sigma_i = \text{numpy.eye}(d)$; where d is dimension

$$\text{e.g. } \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \Sigma_2 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \Sigma_3 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

```
1 sigma = [np.eye(d)]*K
```

```
1 sigma
```

```
[array([[1., 0.],
        [0., 1.]])]
```

$$\Leftrightarrow \text{array} \left[\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right]$$

Step 2) Expectation

- As the first step, evaluate the responsibilities using the current parameter values.
- As the second step, evaluate the indicators for each data point.
- Then, we will see if the data point gets involved in $k=1$, $k=2$, or $k=3$.
- Let us take an example. Let's suppose that we have $X(1000,2)$ dataset.

$$X = \begin{bmatrix} 0.1, -0.3 \\ \vdots \\ -0.6, -0.4 \end{bmatrix} \quad ; 1000 \times 2 \text{ matrix}$$

; Let's calculate Gaussian PDF for $k=1$ case first.

$$\pi_1 = 0.3, \mu_1 = [-3.16, -6.45], \Sigma_1 = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

$$P(X_i | \mu, \Sigma) = \frac{1}{((2\pi)^{D/2} |\Sigma|)^{1/2}} \exp \left[-\frac{1}{2} (X_i - \mu)^T \Sigma^{-1} (X_i - \mu) \right] ; \text{ where } D=2$$

$$\Leftrightarrow P(X_i | \mu_1, \Sigma_1) = \begin{bmatrix} 1.9 \times 10^{-14} \\ \vdots \\ 3.1 \times 10^{-1} \end{bmatrix} ; 1000 \times 1 \text{ matrix}$$

Then,

$$r_1(x) = \frac{\pi_1 \times P(X_i | \mu_1, \Sigma_1)}{\sum_{i=1}^K \pi_i P(X_i | \mu_i, \Sigma_i)} = \frac{1000 \times 1 \text{ matrix}}{1000 \times 1 \text{ matrix}} = \begin{bmatrix} 6.35 \times 10^{-15} \\ \vdots \\ 3.3 \times 10^{-1} \end{bmatrix}$$

$$r_2(x) = 1000 \times 1 \text{ matrix}$$

$$r_3(x) = 1000 \times 1 \text{ matrix}$$

Hence,

$$r = \begin{bmatrix} \dots & \dots & \dots \\ \vdots & \vdots & \vdots \\ \vdots & \vdots & \vdots \end{bmatrix} = [r_1, r_2, r_3] ; 1000 \times 3 \text{ matrix}$$

- For the next step, we're going to estimate the **indicator** for each data point.

Literally, the code will ask to each point 'Does it look like it came from $k=1$, $k=2$, or $k=3$?'

say, we have the first row of $r = [0.1, 0.2, 0.3]$

↳ Indicator will select the third column because of argmax.

```
## Calculate argmax of the responsibility and choose the indicator for each data point properly
for i in range(N):
    I[i] = r[i].argmax()+1 # +1 is needed because of Pythonic characteristic
```

- Finally, $I = \begin{bmatrix} 1 \\ 2 \\ 1 \\ 3 \\ \vdots \end{bmatrix}$ telling us the first data point might come from $k=1$ clustering.

Step 3) Maximization

- Re-estimate the parameters using the current responsibilities.

- In terms of π ,

$$\pi_j = \frac{1}{N} \sum_{n=1}^N r_j(X_n) ; \text{ where } N = \text{number of data point}, j = \text{number of clustering}$$

$$\text{e.g. } \pi_1 = \frac{1}{1000} (r_1(X_1) + r_1(X_2) + r_1(X_3) + \dots + r_1(X_{1000})) = \text{constant}$$

- In terms of μ ,

$$\mu_j = \frac{\sum_{n=1}^N r_j(X_n) \cdot X_n}{\sum_{n=1}^N r_j(X_n)}$$

e.g. $\mu_1 = \frac{A}{\text{constant}} = [X_1, X_2]$; where $A = r_1(X_1) \cdot (X_1, X_2)_1 + r_1(X_2) \cdot (X_1, X_2)_2 + \dots$

- In terms of Σ ,

$$\Sigma_j = \frac{\sum_{n=1}^N r_j(X_n) \cdot (X_n - \mu_j)^T (X_n - \mu_j)}{\sum_{n=1}^N r_j(X_n)} \quad ; \text{ where } \mu_j = \text{new mean (or centroid)}$$

e.g. $\Sigma_1 = \frac{A}{\text{constant}} = [2 \times 2]$; where $A = r_1(X_1) \cdot (X_1, X_2)_1^T \cdot (X_1, X_2)_1 + \dots$

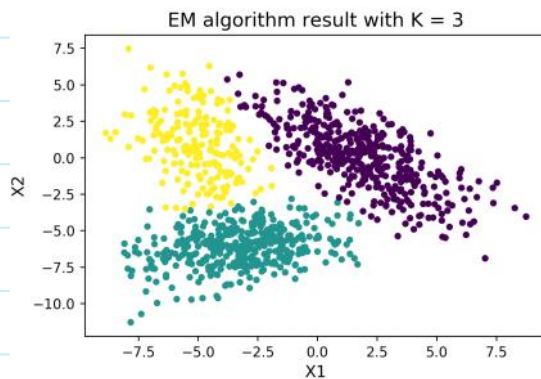
$$= \text{const} \times \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \times [X_1, X_2] + \dots$$

$2 \times 1 \qquad 1 \times 2$

- So, it feels like that we have new parameters.

· Step 4) Repeat the process 2~3 until convergence criteria is satisfied.

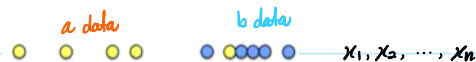
- We will end up getting a kind of result as below ;



Examples

One - Dimensional

· Let's say we have the following observed data ;



· It seems like there are $K=2$ Gaussians with unknown μ and σ^2

· In the end, we will get the result ;

↓
It can be calculated by ; for b data,



$$\mu_b = \frac{X_1 + X_2 + \dots + X_{n,b}}{n_{b,b}} \quad ; \text{ where } n = \text{number of data}$$

$$\sigma_b^2 = \frac{(X_1 - \mu_b)^2 + \dots + (X_n - \mu_b)^2}{n_{b,b}}$$

However, what if we don't know the source?

○ ○ ○ ○ ○ ○ ○ ○ ; no color at all (unobserved data)

- If we know parameters of the Gaussians (μ, σ^2), we'll be able to guess whether point is likely to be *a* or *b* data.

The EM algorithm would be used for this particular type of problem as a probabilistically-grounded way of doing soft clustering.

- Step 1) Randomly place points based on the *k* information ($K = 2$ for this case)

○ ○ ○ ○ ○ ○ ○ ○ ; *x* and *x* are random points

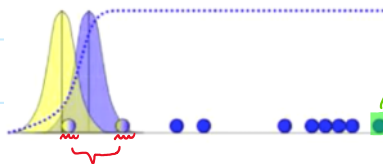
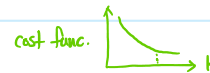
→ Probably, we can get the information from the data set; however, in general, there isn't a great way to choose # of clusters.

- Step 2) An initial guess is made for the parameters (μ, σ^2) and a probability is created on the points.



The elbow method might work for *k*-means clustering such that;

- Step 3) Ask to each point "Does it look like it came from data *a*?"



→ I might ask the data point if it comes from either *a* or *b*.

As can be seen,

- EM algorithm assigns the data to cluster with some probability. (Say 30% of *a* and 70% of *b*)

↳ This is the reason why EM is different with *k*-means algorithm.

• *K*-means algorithm

- Assigned each example to exactly one cluster
- What if clusters are overlapping?
 - Hard to tell which cluster is right
 - Maybe we should try to remain uncertain
- Used Euclidean distance
- What if cluster has a non-circular shape?

k-means : Hard clustering (clusters do not overlap)

EM : Soft clustering (clusters may overlap)

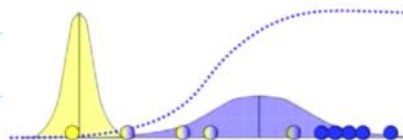
• Gaussian mixture models

- Clusters modeled as Gaussians
- Not just by their mean
- EM algorithm: assign data to cluster with some probability

→ In EM, it can be different distribution types.

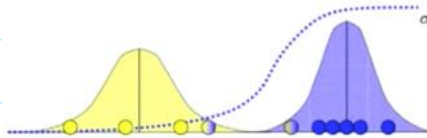
- Step 4) Calculate the parameters based on the newly observed data.

↳ The probability distribution should be tweaked to include the new data, which turns in new shapes.



- Step 5) Repeat the process until convergence criteria has been reached.

e.g. Distributions do not change / Max iteration / ...



; It might be slow on large data set.

; Hence, it may reveal hidden patterns. Note that it won't work well if clusters contain few points.

Two-dimensional

