NAME: - Medhansh K PRN: - 122A8035 BATCH:- B2/TE-AIDS

EXPERIMENT 9

**AIM: -** Expectation-Maximization Algorithm

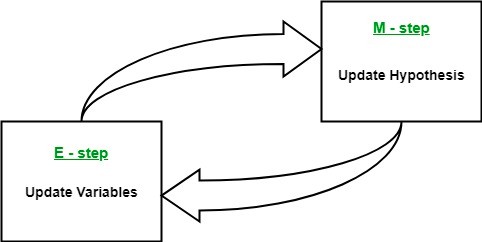
# THEROY: -

The Expectation-Maximization (EM) algorithm is an iterative method used in [unsupervised](https://www.geeksforgeeks.org/unsupervised-learning/) [machine learning](https://www.geeksforgeeks.org/unsupervised-learning/) to estimate unknown parameters in statistical models. It helps find the best values for unknown parameters, especially when some data is missing or hidden.

It works in two steps:

1. E-step (Expectation Step): Estimates missing or hidden values using current parameter estimates.
2. M-step (Maximization Step): Updates model parameters to maximize the likelihood based on the estimated values from the E-step.

This process repeats until the model reaches a stable solution, improving accuracy with each iteration. EM is widely used in clustering (e.g., Gaussian Mixture Models) and handling missing data.

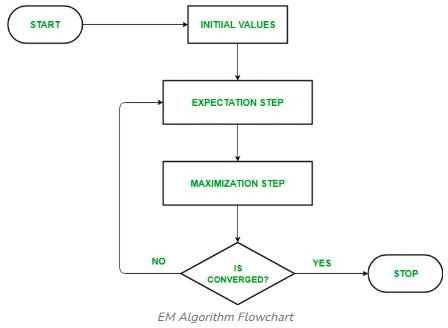


By iteratively repeating these steps, the EM algorithm seeks to maximize the likelihood of the observed data. It is commonly used for clustering, where latent variables are inferred and has

applications in various fields, including machine learning, computer vision, and natural language processing.

# How Expectation-Maximization (EM) Algorithm Works:

So far, we’ve discussed the key terms in the EM algorithm. Now, let’s dive into how the EM algorithm works. Here’s a step-by-step breakdown of the process:



# Initialization:

The algorithm starts with initial parameter values and assumes the observed data comes from a specific model.

# E-Step (Expectation Step):

* Estimate the missing or hidden data based on the current parameters.
* Calculate the posterior probability (responsibility) of each latent variable given the observed data.
* Compute the log-likelihood of the observed data using the current parameter estimates.

# M-Step (Maximization Step):

* Update the model parameters by maximizing the log-likelihood computed in the E-step.
* This involves solving an optimization problem to find parameter values that improve the model fit.

sns.kdeplot(X) plt.xlabel('X') plt.ylabel('Density') plt.title('Density

plt.show()

Estimation

of

X')

# Convergence:

* Check if the model parameters are stable (converging).
* If the changes in log-likelihood or parameters are below a set threshold, stop. If not, repeat the E-step and M-step until convergence is reached.

**Expectation-Maximization Algorithm Step by Step Implementation Step 01 : Import the necessary libraries**

**import import from from**

**import**

**numpy seaborn**

**scipy.stats scipy.stats**

**matplotlib.pyplot**

**as as import**

**import**

**as**

**np sns** norm

gaussian\_kde

**plt**

**Step 02 : Generate a dataset with two Gaussian components**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| mu1, | sigma1 | | = | 2, | 1 |
| mu2, | sigma2 | | = | -1, | 0.8 |
| X1 | = | np.random.normal(mu1, | | sigma1, | size=200) |
| X2 | = | np.random.normal(mu2, | | sigma2, | size=600) |
| X | = | | np.concatenate([X1, | | X2]) |

**Step 03: Initialize parameters**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| mu1\_hat, |  | sigma1\_hat |  | = |  | np.mean(X1), |  | np.std(X1) |
| mu2\_hat, |  | sigma2\_hat |  | = |  | np.mean(X2), |  | np.std(X2) |
| pi1\_hat, | pi2\_hat | = | len(X1) |  | / | len(X), len(X2) | / | len(X) |

**Step 04: Perform EM algorithm**

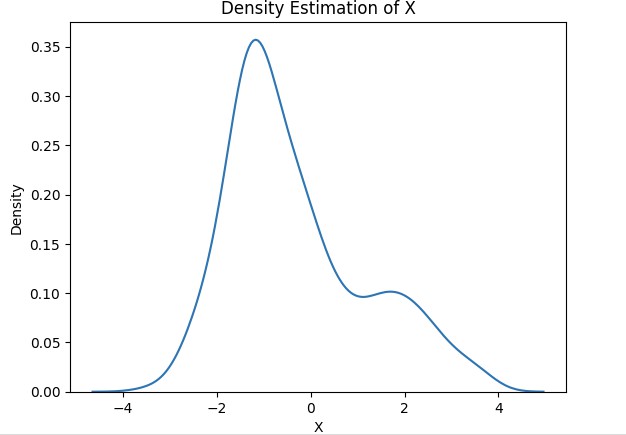
|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| num\_epochs = 20  log\_likelihoods = [] | | | | | | | | | | |
| **for**  *#*  gamma1 | = | | epoch  *E-step:*  pi1\_hat | **in**  *Compute*  \* norm.pdf(X, | | | | range(num\_epochs):  *responsibilities*  mu1\_hat, sigma1\_hat) | | |
| gamma2 total gamma1 gamma2 | = | | pi2\_hat  = | \* norm.pdf(X, gamma1  /=  /= | | | | mu2\_hat, sigma2\_hat)  + gamma2  total total | | |
| *#*  mu1\_hat |  | = | *M-step:*  np.sum(gamma1 | | \* | | *Update*  X) / | |  | *parameters*  np.sum(gamma1) |
| mu2\_hat  sigma1\_hat | = | = | np.sum(gamma2  np.sqrt(np.sum(gamma1 | | \*  \* (X | | X) /  - mu1\_hat)\*\*2) | | / | np.sum(gamma2)  np.sum(gamma1)) |
| sigma2\_hat pi1\_hat pi2\_hat | = |  | np.sqrt(np.sum(gamma2  =  = | | \* (X | | - mu2\_hat)\*\*2) | | / | np.sum(gamma2)) np.mean(gamma1) np.mean(gamma2) |
| *#*  log\_likelihood | *Compute*  = np.sum(np.log(pi1\_hat | | | | | \* norm.pdf(X, | | *log-likelihood*  mu1\_hat, sigma1\_hat) | | |
| + pi2\_hat \* norm.pdf(X, mu2\_hat, sigma2\_hat)))  log\_likelihoods.append(log\_likelihood) | | | | | | | | | | |

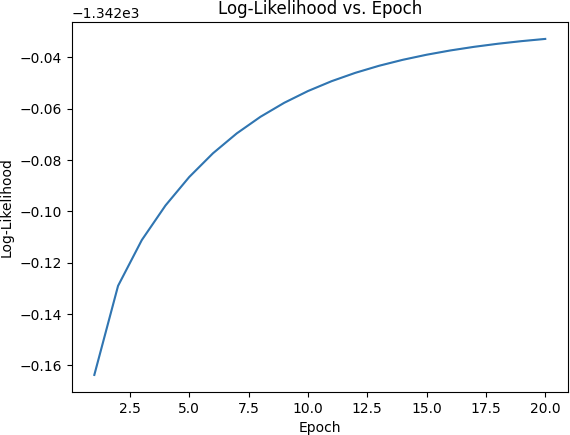
|  |  |  |
| --- | --- | --- |
| plt.plot(range(1, | num\_epochs+1), | log\_likelihoods) |
| plt.xlabel('Epoch') |  |  |
| plt.ylabel('Log-Likelihood') |  |  |
| plt.title('Log-Likelihood | vs. | Epoch') |
| plt.show() |  |  |

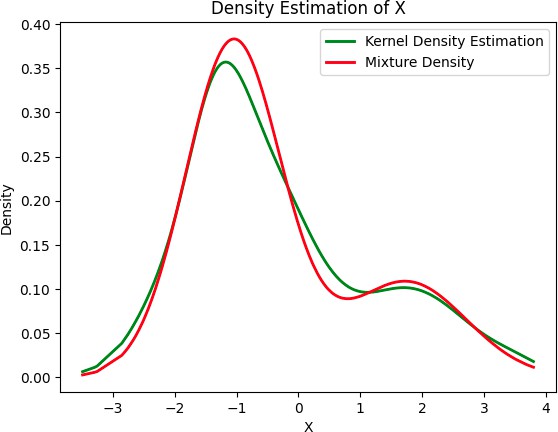
**Step 05: Plot the final estimated density**

|  |  |  |
| --- | --- | --- |
| X\_sorted = np.sort(X) density\_estimation = pi1\_hat\*norm.pdf(X\_sorted,  mu1\_hat,  sigma1\_hat) + pi2\_hat \* norm.pdf(X\_sorted, mu2\_hat,  sigma2\_hat) | | |
| plt.plot(X\_sorted, | gaussian\_kde(X\_sorted)(X\_sorted), color='green', | linewidth=2) |
| plt.plot(X\_sorted, | density\_estimation, color='red', | linewidth=2) |
| plt.xlabel('X') |  |  |
| plt.ylabel('Density') |  |  |
| plt.title('Density | Estimation of | X') |
| plt.legend(['Kernel | Density Estimation','Mixture | Density']) |
| plt.show() |  |  |

**OUTPUT: -**

****



****

**CONCLUSION: -**

The EM algorithm iteratively estimates missing data and updates model parameters to improve accuracy. By alternating between the E-step and M-step, it refines the model until it converges, making it a powerful tool for handling hidden or incomplete data in machine learning.