Machine Intelligence and Learning

ELL409

Assignment 2

Vivek Muskan

2017MT10755

Topics

1. Expectation Maximization clustering of Gaussian Mixture Model
2. Lasso Regression
3. Fuzzy C-Means Clustering

Expectation Maximization Clustering of Gaussian Mixture Model

We assume that the distribution of data can be represented as a convex linear combination of Normal Distributions with different parameters.

$$\mathbf{p(x)} = \sum_{\mathbf{k=1}}^{\mathbf{K}} \pi_{\mathbf{k}} \mathcal{N}(\mathbf{x} \mid \mu_{\mathbf{k}}, \Sigma_{\mathbf{k}})$$

Number of Gaussians

Mixing coefficient: weightage for each Gaussian dist.

Algorithm :

Initialize all parameters randomly.

Calculate the expectations of each parameter.

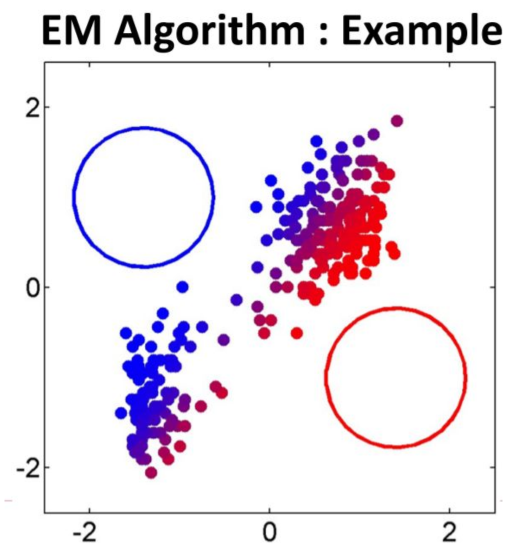
$$\gamma_j(\mathbf{x}) = \frac{\pi_k \mathcal{N}(\mathbf{x} \mid \mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x} \mid \mu_j, \Sigma_j)}$$

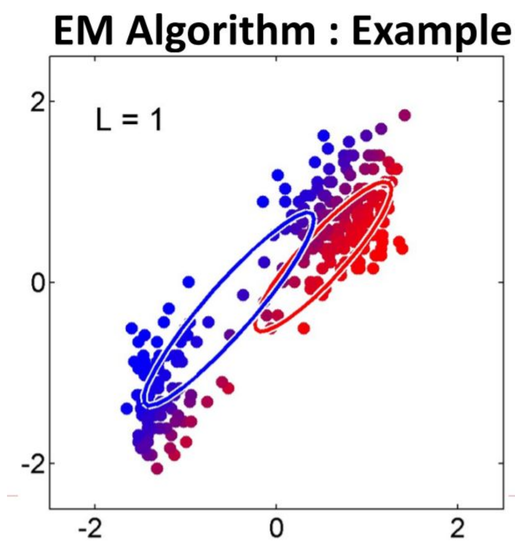
Then Using this expectation, we update the parameters to maximize expectation.

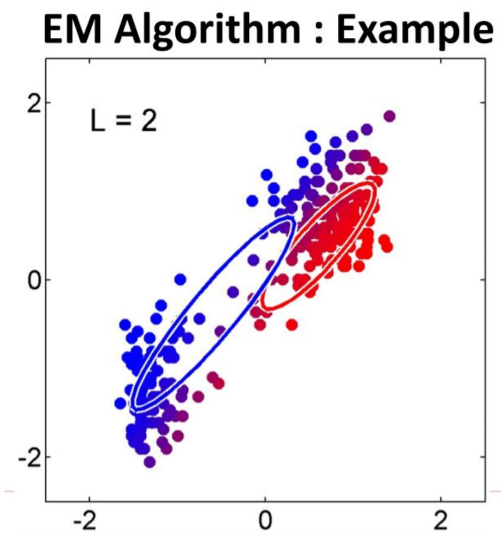
$$\mu_j = \frac{\sum_{n=1}^{N} \gamma_j(\mathbf{x}_n)\mathbf{x}_n}{\sum_{n=1}^{N} \gamma_j(\mathbf{x}_n)} \qquad \Sigma_j = \frac{\sum_{n=1}^{N} \gamma_j(\mathbf{x}_n)(\mathbf{x}_n - \mu_j)(\mathbf{x}_n - \mu_j)^T}{\sum_{n=1}^{N} \gamma_j(\mathbf{x}_n)} \qquad \pi_j = \frac{1}{N}\sum_{n=1}^{N} \gamma_j(\mathbf{x}_n)$$

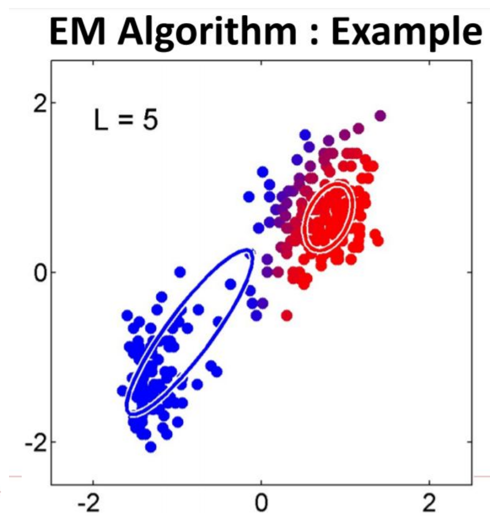
I simply applied these algorithms to determine k-possible clusters as ‘k’ different Normal Distribution parameters.
Here is a pictorial view of what happens in multiple epochs. The termination of this algorithm is governed by the convergence of log likelihood function of the distribution.

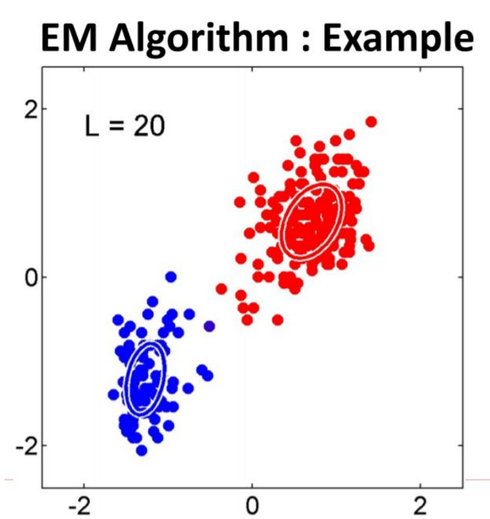
```python
def EM_Algo(x,k,delta):

  col = x.shape[1]
  #intialize mu, sigma, pi
  mu = np.random.randn(k,col)
  #sigma = abs(np.random.randn(k,col,col))
  sigma = abs(np.array([p*np.eye(col) for p in np.random.randn(k)]))
  pi = np.random.randn(k)
  err = delta + 1

  # Making Cov matrix Invertible
  for i in range(k):
    det = np.linalg.det(sigma[i])
    #print(det)
    while det==0:
      sigma[i] += 0.01*eye(col)
      det = np.linalg.det(sigma[i])
      #print(det)

  # Calculate log likelihood
  loghood_old = logHood(x,mu,pi,sigma)

  epoch = 1
  while(err > delta):
    print("Epoch No -> ",epoch)
    # Update Step
    for j in range(k):
      print("   Column No : ",j)
      # mu update
      num_mu = np.zeros(mu[0].shape)
      for t in x: num_mu+= Gamma(pi, mu, sigma, t, j)*t
      den = 0
      for t in x: den+= Gamma(pi, mu, sigma, t, j)

      # Sigma Update
      num_sig = np.zeros(sigma[0].shape)
      for t in x: num_sig+= Gamma(pi, mu, sigma, t, j)*np.matmul((t-mu[j]),(t-mu[j].transpose()))

      # Making Cov matrix Invertible
    for i in range(k):
      det = np.linalg.det(sigma[i])
      #print(det)
      while det==0:
        sigma[i] += 0.01*np.eye(col)
        det = np.linalg.det(sigma[i])
        #print(det)

      # Pi Update
      pi[j] = (1/(len(x)))*den
      sigma[j] = num_sig/den
      mu[j] = num_mu/den

    # Making Cov matrix Invertible
    for i in range(k):
      det = np.linalg.det(sigma[i])
      #print(det)
      while det==0:
        sigma[i] += 0.01*np.eye(col)
        det = np.linalg.det(sigma[i])
        #print(det)

    loghood_new = logHood(x,mu,pi,sigma)
    err = loghood_new - loghood_old
    loghood_old = loghood_new
    epoch+=1

  return mu, sigma, pi
```











My method simply returns all parameters of distributions.

2. Lasso Regression

Lasso regression is simply L1-Norm regularization of the linear regression. Since the L1 norm is not differentiable everywhere so we can’t use Gradient Descent Method directly on the Loss function :

$$
\begin{aligned}
\mathcal{L}(\boldsymbol{\beta};\lambda) &= \|\mathbf{Y} - \mathbf{X}\,\boldsymbol{\beta}\|_2^2 + \lambda_1\|\boldsymbol{\beta}\|_1 \\
&= \underbrace{\sum_{i=1}^{n}(Y_i - \mathbf{X}_{i*}\,\boldsymbol{\beta})^2}_{\text{sum of squares}} + \lambda_1 \underbrace{\sum_{j=1}^{p}|\beta_j|}_{\text{lasso penalty}}
\end{aligned}
$$

So we will be solving this as minimization problem on the loss function of Linear Regression with a constraint on weight of Beta having L1-Norm less than Lambda, which is precisely known as Coordinate Gradient Descent Method.

Here is the main updating step of this algo:

Coordinate descent update rule:

Repeat until convergence or max number of iterations:

- For $j = 0, 1, \ldots, n$
- Compute $\rho_j = \sum_{i=1}^{m} x_j^{(i)}(y^{(i)} - \sum_{k \neq j}^{n} \theta_k x_k^{(i)}) = \sum_{i=1}^{m} x_j^{(i)}(y^{(i)} - \hat{y}_{pred}^{(i)} + \theta_j x_j^{(i)})$
- Set $\theta_j = S(\rho_j, \lambda)$
- Note that if there is a constant term, then it is not regularized so $\theta_0 = \rho_0$

I used the same method to update the weights or Beta.

```python
def coordinate_descent_lasso(theta,X,y,lamda = .01, num_iters=100, intercept = False):
    '''Coordinate gradient descent for lasso regression - for normalized data.
    The intercept parameter allows to specify whether or not we regularize theta_0'''

    #Initialisation of useful values
    m,n = X.shape
    X = X / (np.linalg.norm(X,axis = 0)) #normalizing X in case it was not done before

    #Looping until max number of iterations
    for i in range(num_iters):

        #Looping through each coordinate
        for j in range(n):

            #Vectorized implementation
            X_j = X[:,j].reshape(-1,1)
            y_pred = X @ theta
            rho = X_j.T @ (y - y_pred  + theta[j]*X_j)

            #Checking intercept parameter
            if intercept == True:
                if j == 0:
                    theta[j] =  rho
                else:
                    theta[j] =  soft_threshold(rho, lamda)

            if intercept == False:
                theta[j] =  soft_threshold(rho, lamda)

    return theta.flatten()
```

3. Fuzzy C-Means Clustering

Fuzzy C means clustering is the extension of K-means clustering concept from Crisp Set into Fuzzy sets. Here also we select K random Centres (Neighbours) and then rest of all points have to be assigned a membership value w.r.t each Centres instead of assigning it to the one of the centre based on some distance function.

I have used Euclidean distance in my project. I used sklearn distance library.
Here is the algorithm that I used. This is the objective function or Loss Function (L2-Norm)

$$J(U,V) = \sum_{i=1}^{n} \sum_{j=1}^{c} (\mu_{ij})^m \left\| \mathbf{x}_i - \mathbf{v}_j \right\|^2$$

<u>Algorithm</u> :

1) Randomly select 'c' cluster centers.

2) Calculate the fuzzy membership 'μ_{ij}' using:

$$\mu_{ij} = 1 / \sum_{k=1}^{c} (d_{ij} / d_{ik})^{(2/m-1)}$$

3) Compute the fuzzy centers 'v_j' using:

$$v_j = (\sum_{i=1}^{n} (\mu_{ij})^m x_i) / (\sum_{i=1}^{n} (\mu_{ij})^m), \forall j = 1, 2, ...c$$

4) Repeat step 2) and 3) until the minimum 'J' value is achieved or $||U^{(k+1)} - U^{(k)}|| < \beta$.

```python
def fcm(data, n_clusters=1, n_init=30, m=2, max_iter=300, tol=1e-16):

    for iter_init in range(n_init):

        # Randomly initialize centers
        centers = data[np.random.choice(
            data.shape[0], size=n_clusters, replace=False
            ), :]

        # Compute initial distances
        # Zeros are replaced by eps to avoid division issues
        dist = np.fmax(
            cdist(centers, data, metric='sqeuclidean'),
            np.finfo(np.float64).eps
        )

        for iter1 in range(max_iter):

            # Compute memberships
            u = (1 / dist) ** (1 / (m-1))
            um = (u / u.sum(axis=0))**m

            # Recompute centers
            prev_centers = centers
            centers = um.dot(data) / um.sum(axis=1)[:, None]

            dist = cdist(centers, data, metric='sqeuclidean')

            if np.linalg.norm(centers - prev_centers) < tol:
                break

    return centers
```

When we give data as input in this function it gives out k centres which exactly defines the centre of clusters. Clusters can be obtained by either defuzzification of these centres or by simply taking Sup-Norm over all Centres.

References

1. EM Clustering of GMM : http://www.cse.iitm.ac.in/~vplab/courses/DVP/PDF/gmm.pdf
2. Lasso Regression :
   https://stats.stackexchange.com/questions/123672/coordinate-descent-soft-thresholding-update-operator-for-lasso?noredirect=1&lq=1
   https://xavierbourretsicotte.github.io/lasso_implementation.html
3. Fuzzy C Means :
   https://home.deib.polimi.it/matteucc/Clustering/tutorial_html/cmeans.html
   https://sites.google.com/site/dataclusteringalgorithms/fuzzy-c-means-clustering-algorithm