Classification: K-means, and the EM algorithm

1. We consider a mixture model, with K components, where data points X_i, $i = 1,...,n$ have a probability p_k to be in component $k : P(Z_i = k) = p_k$, and, conditional on $Z_i = k, X_i \sim N(\mu_k, D_k)$, a multivariate Gaussian distribution with mean μ_k, and diagonal (not full) covariance matrix D_k.

   We consider $z_i^k \in (0,1), z_i^k = 1$ if $Z_i = k$ and $z_i^k = 0$ otherwise.The Complete log-likelihood at iteration t can be written as :

$$l_{c,t} = logp(X,Z) = \sum_{i=1}^{n} log(p(X_i|Z_i)p(Z_i)) = \sum_{i=1}^{n}\sum_{j=1}^{K} z_i^j log(p_{j,t}) + z_i^j log(N(X_i|\mu_{j,t}, D_{j,t}))$$

E-Step

We can now write the expectation of the previous quantity with respect to the conditional distribution of Z given X :

$$\mathbb{E}_{Z|X}(l_{c,t}) = \sum_{i=1}^{n}\sum_{j=1}^{K} T_i^j log(p_{j,t}) + T_i^j log(N(X_i|\mu_{j,t}, D_{j,t}))$$

where $T_i^j = p(Z_i = j|X_i) = \frac{p(X_i|Z_i=j)p(Z_i=j)}{p(X_i)} = \frac{p_j N(X_i|\mu_{j,t}, D_{j,t})}{\sum_{j'=1}^{K} p_{j'} N(X_i|\mu_{j',t}, D_{j',t})}$ (From Data and previous estimates)

M-Step

For the M-step, we this need to maximize:

$$\sum_{i=1}^{n}\sum_{j=1}^{K} T_i^j log(p_{j,t}) + T_i^j \left[log(\frac{1}{(2\pi)^{K/2}}) + log(\frac{1}{|D_{j,t}|^{\frac{1}{2}}}) - \frac{1}{2}(X_i - \mu_{j,t})^T D_{j,t}^{-1}(X_i - \mu_{j,t}) \right]$$

As the sum is separated into two terms independent along the variables we can first maximize with respect to $P_t = (p_1, ..., p_K)$:

$$\max_{P_t} Q(P_t) = \sum_{i=1}^{n}\sum_{j=1}^{K} T_i^j log(p_{j,t})$$

maximizing Q can be solved by solving the problem $\begin{cases} \max Q(P_t) \\ \text{under constraint :} \\ \sum p_j = 1 \end{cases}$

we consider the Lagrange multiplier : $h(P_t) = Q(P_t) + \lambda(1 - \sum_{j=1}^{K} p_j)$

$$\frac{\partial h}{\partial p_j} = \frac{1}{p_j}\sum_{i=1}^{n} T_i^j - \lambda = 0 \implies \sum_{j=1}^{K} p_j\lambda = \sum_{j=1}^{K}\sum_{i=1}^{n} T_i^j = \sum_{i=1}^{n}\sum_{j=1}^{K} T_i^j \text{since } \sum_{j=1}^{K} p_j = 1 \quad \text{and } \sum_{j=1}^{K} T_i^j = 1 \implies \lambda = n$$

and finally

$$p_{j,t+1} = \frac{\sum_{i=1}^{n} T_i^j}{n}$$

We now do the same for $H(\mu, D) = \sum_{i=1}^{n}\sum_{j=1}^{K} T_i^j \left[log(\frac{1}{(2\pi)^{K/2}}) + log(\frac{1}{|D_{j,t}|^{\frac{1}{2}}}) - \frac{1}{2}(X_i - \mu_{j,t})^T D_{j,t}^{-1}(X_i - \mu_{j,t}) \right]$ by maximizing over D_j^{-1} and μ_j and since D is diagonal this gives us easier expressions for the Update of our

estimates: $D_j = (d_j^1, .., d_j^q)$ in particular : $log(\frac{1}{|D_{j,t}|^{\frac{1}{2}}}) = \frac{1}{2}\sum_{m=1}^{q} log(1/d_j^m)$

$\nabla_{\mu_j}(H) = \sum_{i=1}^{n} T_i^j D_{j,t}^{-1}(X_i - \mu_{j,t})$ and $\nabla_{\mu_j}(H) = 0 \implies \sum_{i=1}^{n} T_i^j (X_i - \mu_{j,t+1}) = 0$

so finally :

$$\mu_{j,t+1} = \frac{\sum_{i=1}^{n} T_i^j X_i}{\sum_{i=1}^{n} T_i^j}$$

and for D_j:

$\nabla_{D_j^{-1}}(H) = \sum_{i=1}^{n} T_i^j \left[\frac{1}{2} D_j - \frac{1}{2} * diag\left((X_i - \mu_{j,t})(X_i - \mu_{j,t})^T\right)\right]$ and by setting the derivative to 0 we get :

$$D_{j,t+1} = \frac{\sum_{i=1}^{n} T_i^j * diag\left((X_i - \mu_{j,t})(X_i - \mu_{j,t})^T\right)}{\sum_{i=1}^{n} T_i^j}$$

2. The Model is Implemented on python : Let's say we have a feature vector with many dimensions. The covariance matrix would have many parameters and the more parameters, the more data we need to estimate them. In the case of a diagonal matrix we have way less parameters to estimate. This is why we like having uncorrelated features!

3. We can see projections of each Couple of features along with their True Labels and The ellipsis showing the probability of a data-point belonging to the represented Gaussian Model.

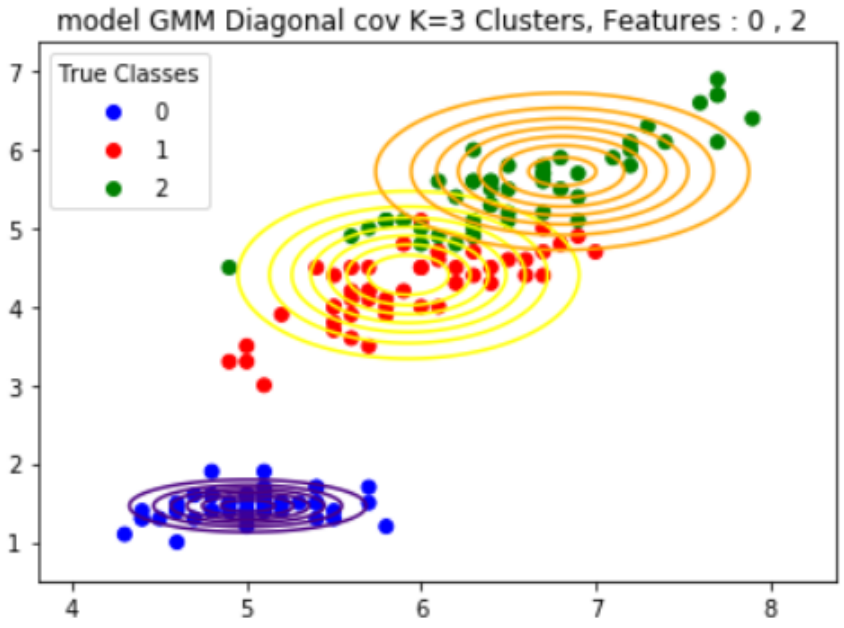

Figure 1: GMM model with a Diagonal Covariance matrix for K=3 clusters and for Features pair 0,2 .

In Figure 1 we can see the impact of using a diagonal covariance matrix when looking at the direction of the ellipses as they all have the same directions which are the same as the axis we're plotting on.

We can also see these directions when comparing this model to the full covariance matrix model. As we can see in Figure 2 the ellipses now have their own directions which gives them more freedom to fit to data for example

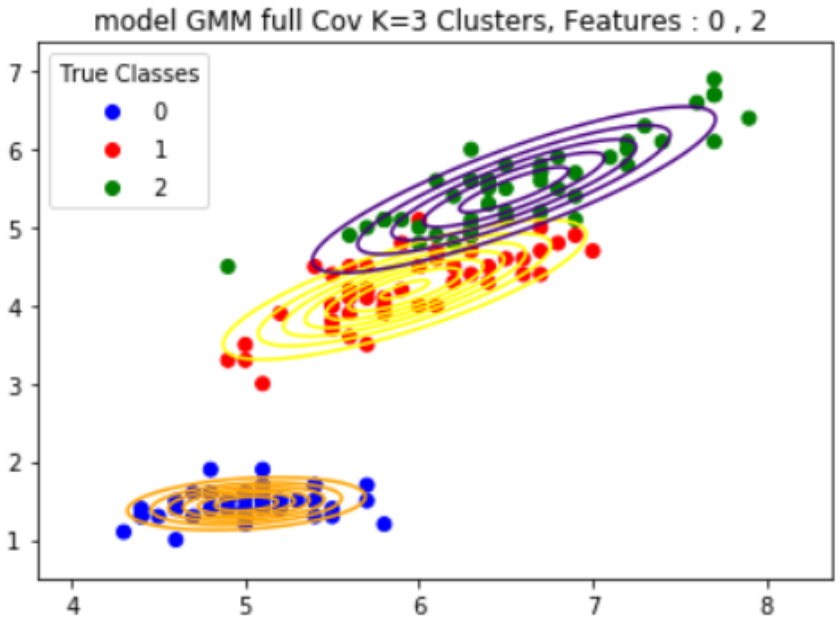

Figure 2: GMM model with a Full Covariance matrix for K=3 clusters and for Features pair 0,2 .

when we compare the performance of the models for classes 2 and 1 and for the Features we're showing here we can see that the full covariance model can have a better performance.

Finally We can Compare the GMM model to the Clusters provided by K Means Algorithm as we can see the Clusters Given by the Latter and the Ellipsis showing the output of the former

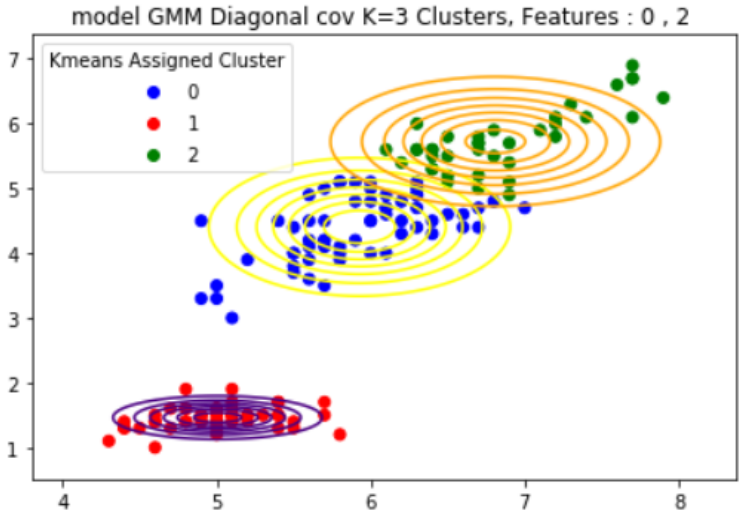

Figure 3: GMM model with a Diagonal Covariance matrix and Kmeans Clusters for K=3 clusters and for Features pair 0,2 .

The Clusters Given By the two models seems to be quite similar even though the Gaussian Mixture Model is

not really discriminative in the sense that it gives probabilities of belonging to a Cluster rather than a rigid class assignment.

4. K-means will work pretty well when the width of the different clusters are similar, for example if we deal with spheres. But clustering by K-means could also be disappointing in some cases such as the example given in our code and shown in Figure 4

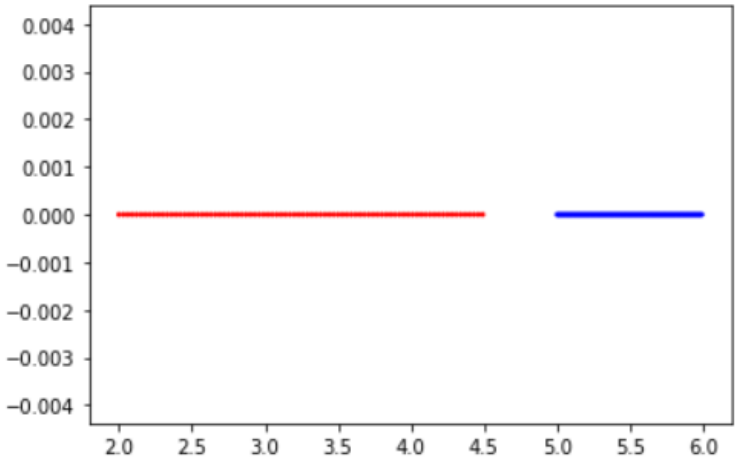

Figure 4: Unbalanced 1 Dimensional Dataset with two classes

If we run Kmeans on this dataset it will yield the wrong classes as shown in figure 5

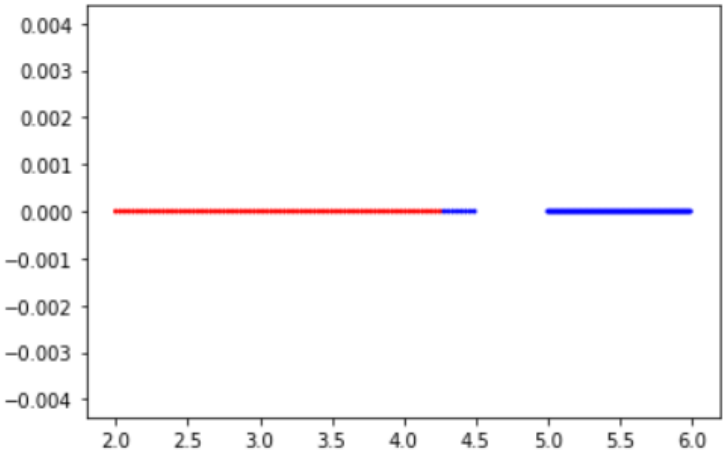

Figure 5: Kmeans Clusters Over the Unbalanced Dataset

Notice How Kmeans gets a part of the first cluster wrong However running our Digonal Gaussian Mixture Model, this yields the results show in Figure 6 which shows how the GMM model outperforms the Kmeans model in this case

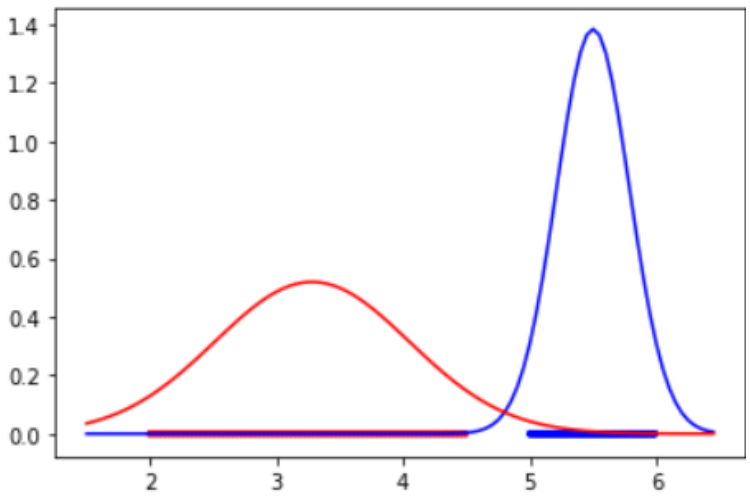

Figure 6: GMM Probabiliy Density Functions Over the Unbalanced Dataset

Hence We can see how this issue with Kmeans is solvet by the GMM model.

Graphs, algorithms and Ising

Undirected Chain model:
We have:

$$p(x) = \frac{1}{Z}\prod_{i=1}^{n}\psi_i(x_i)\prod_{i=1}^{n-1}\psi_{i,i+1}(x_i, x_{i+1})$$

To compute marginal probabilities, we need first to compute descending and ascending messages, which are, as seen in the course:

Descending messages:

$$\mu_{i->i-1}(x_{i-1}) = \sum_{x_i}\psi_{i-1,i}(x_{i-1}, x_i) * \psi_i(x_i) * \mu_{i+1->i}(x_i)$$

Ascending messages:

$$\mu_{i->i+1}(x_{i+1}) = \sum_{x_i}\psi_{i,i+1}(x_i, x_{i+1}) * \psi_i(x_i) * \mu_{i-1->i}(x_i)$$

With this we can propagate backward and forward messages. We then use this to compute marginal probabilities and Z:

$$p(x_i) = \frac{1}{Z}\mu_{i-1->i}(x_i) * \mu_{i+1->i}(x_i) * \psi_i(x_i)$$

$$1 = \sum_{x_i} p(x_i) \quad => \quad Z = \sum_{x_i}\mu_{i-1->i}(x_i) * \mu_{i+1->i}(x_i) * \psi_i(x_i)$$

The function Ψ_i could be represented as a Vector. In the case where X_i takes discrete values in 1..K, Ψ_i could be represented as $[\Psi_i(X_i = j); j = 1 :: K]$. We can generalize this to the continuous case where we sample discrete values from our continuous interval. Following the same logic, the edges functions $\Psi_{i-1,i}$ would be represented as a matrix for all the possible discrete values of X_i and X_{i-1} $[[\Psi_{i,i-1}(X_i = j, Xi - 1 = l); j = 1 :: K]; l = 1 :: K']$ To transform the Ising model into an undirected chain model, we can construct supernodes on one dimension of the model. In our case the width is smaller than the height, so we will consider a junction tree that assembles all the nodes on one line of height into a super node. The supernode will contain 2^10 possible states. With this model the probability factorizes as follows:

$$p(x) = \frac{1}{Z} \prod_{h_i} \psi_{h_i}(x_{h_i}) \prod_{h_i} h_i \psi_{h_i,h_i+1}(x_{h_i}, x_{h_i+1})$$

Where:

$$\psi_i(x_{h_i}) = \prod_{k=1}^{w} e^{\alpha x_{i,k}} \prod_{k=1}^{w-1} e^{\beta I_{x_{i,k}=x_{i,k+1}}}$$

And:

$$\psi_{h_i,h_i+1}(x_{h_i}, x_{h_i+1}) = \prod_{k=1}^{w} e^{\beta I_{x_{i,k}=x_{i+1,k}}}$$

Result:

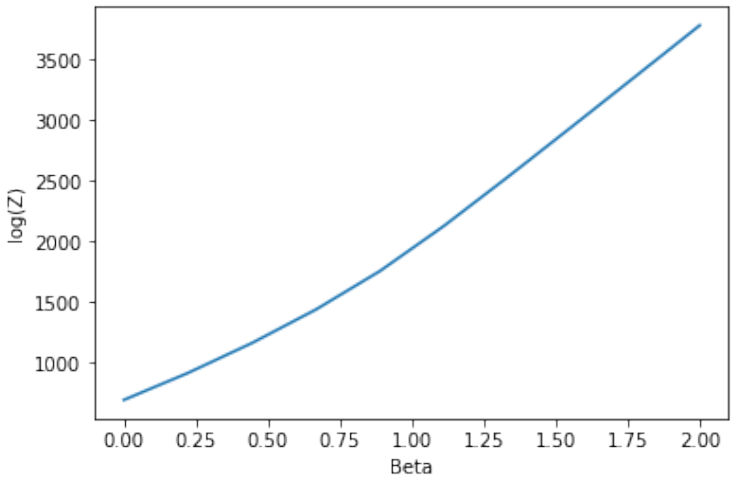

Figure 7: Normalization factor Z of the Ising Model

For the Loopy Belief Propagation algorithm, we compute the Bethe approximation of the partition function and the marginal distribution of each node with the message passing method:

1. Initialization:

For all i,j neighbours, we initialize the message from i to j as:

$$\mu^0_{i->j}(x_j) = 1$$

for all x_j

2. Message passing:

We iterate over t the following recursive relation:

$$\mu^{t+1}_{i->j}(x_j) = \frac{1}{Z_B} \sum_{x_i} \psi_{i->j}(x_i, x_j) \prod_k \mu^t_{k->i}(x_i)$$

We repeat this until convergence to $\mu^*_{i->j}$

3. Approximated Marginal Probabilities:

$$b_i(x_i) = \frac{1}{Z_B} \prod_{N(i)} \mu^*_{j->i}(x_i)$$

$$b_{i,j}(x_i, x_j) = \frac{1}{Z_B} \psi_{i,j}(x_i, x_j) \prod_{N(j)-i} \mu^*_{k->j}(x_j) \prod_{N(i)-j} \mu^*_{k->i}(x_i)$$