Gaussian Mixture Modelling for Clustering
CS342: Homework Assignment

Torben Berndt, U2062238

December 6, 2021

1 Introduction

This report focuses on implementing and testing a clustering approach based on Gaussian Mixture Modelling (GMM) and the Expectation-Maximization (EM) algorithm. Similar to soft *k-means*, GMM offers a probabilistic approach to modelling data by allowing for soft assignments to clusters. However, unlike soft *k-means*, which only considers the mean of clusters, GMM also accounts for their variance.

In this project, we use the Iris Dataset from the UCI Machine Learning Repository. More information can be found on their website or the coursework specifications.

2 Data Visualisation (II.B)

First, we want to visualise the data-points of the Iris Dataset in 2 dimensions. Using Principle Component Analysis (PCA) we reduce the dimension of the feature vectors from four to two.

Figure 1 shows the samples of the Iris Dataset projected onto their top two principal components. Their colour corresponds to their class, in accordance with the legend.

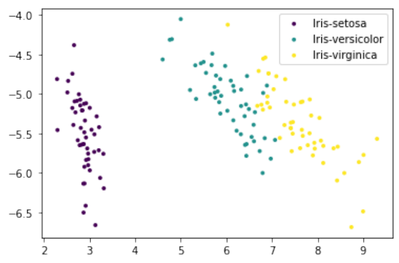

Figure 1: Iris Dataset in 2D with true labels.

3 Clustering using GMM and the EM algorithm (II.C)

In this section, we seek to implement a GMM model using the EM algorithm. We implement a class with the following structure:

```
class GMM_EM():
    '''Gaussian Mixture Modelling with the E-M-algorithm'''
    def __init__(self, num_components, num_samples, tol):
        '''Initialise variables'''
        return None
    def fit(self, X, init_centres):
        '''Train the GMM model using the EM algorithm'''
        return None
    def predict(self, X, tol):
        '''Use the trained model to hard-assign each point in X'''
        return y_hat
    def misclass_accuracy(self, X, y):
        '''Calculate misclassifications and accuracy'''
        return misclass, acc
    def calc_resp(self, X, sig, mu, pi, k):
        '''Calculate responsibilities for given paramters'''
        return r_i_c
    def obj_fun(self, resp, X, mu, sig):
        '''Calculate the objective function for GMM'''
        return obj
```

3.1 Finding initial parameters

First, we a need set of initial parameters $\theta^0 = \{\pi_c^0, \boldsymbol{\mu}_c^0, \Sigma_c^0\}_{c=1}^k$. To find them we use sklearn's implementation of the *k-means* algorithm to cluster the newly obtained data points into $k = 3$ clusters.

Figure 2 below shows the 2D data points again, this time coloured according to their predicted label by the *k-means* model. It also displays the circular shape of the clusters using patches centred at the cluster centres (although the shape may not appear to be circular due to the scaling of the axes). Their radii are the distances from each cluster centre to the most distant point belonging to that cluster.

As mentioned before, *k-means* assumes the variance of the distribution of each dimension is the same. Hence, along with the obtained cluster centres $\{\boldsymbol{\mu}_c^0\}_{c=1}^k$ we use initial covariance matrices $\{\Sigma_c^0\}_{c=1}^k = \{I_2\}_{c=1}^k$ where I_2 is the 2×2 identity matrix, to calculate our initial responsibilities. Since the prior $p(z_i = c|\Theta) = \frac{1}{k}$, assuming a uniform assignment to clusters, the formula for calculating responsibilities simplifies to

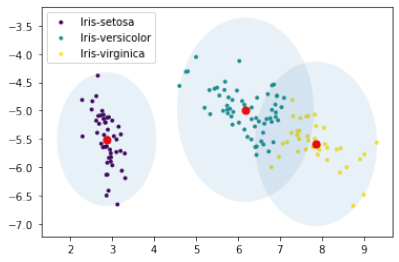

Figure 2: Iris Dataset in 2D with predicted labels (k-means).

$$\begin{aligned} r_c^i &= p(z_i = c|\boldsymbol{x}_i, \Theta) \\ &= \frac{p(\boldsymbol{x}_i|z_i = c, \Theta)}{\sum_{c'=1}^k p(\boldsymbol{x}_i|z_i = c', \Theta)} \end{aligned}$$

where the $\boldsymbol{x}_i$ initially follow a multivariate mixed normal distribution with means $\{\boldsymbol{\mu}_c^0\}_{c=1}^k$ and covariances $\{\Sigma_c^0\}_{c=1}^k$, i.e. $p(\boldsymbol{x}_i|z_i = c, \Theta) = p(\boldsymbol{x}_i|\boldsymbol{\mu}_c, \Sigma_c)$.

3.2 Implementation of the EM algorithm

Our implementation of the EM algorithm has the following structure:

```
class GMM_EM():
    def fit(self, X, init_centres, tol):
        '''Train the GMM model using the EM algorithm'''
        ->calculate the initial parameters
        while (algorithm has not converged):
            #E step
            ->calculate responsibilities for current
              parameters
            #M step
            ->update parameters
```

The crucial part here is the convergence of the EM algorithm. Following an idea similar to the one presented in the lecture notes for *k-means*, we say the algorithm has converged if the last iteration of the algorithm leaves the assignments of the samples unchanged, i.e. the responsibilities of the previous iteration equal the updated ones. Since in practice this does not happen or would take too long to compute, we implement a tolerance *gmm.tol* and say the algorithm has converged if the Euclidean distance between responsibilities of successive iterations differ by less than *gmm.tol*, i.e.

$$\|\boldsymbol{r^i} - \boldsymbol{r^{i-1}}\| < gmm.tol.$$

Choosing a value for *gmm.tol* is a trade-off between the accuracy of the model and computational cost.

In a hyperparameter-like approach we attempt to find a good value for *gmm.tol* by calculating the accuracy of the model for different values of *gmm.tol* (see 3.3 for our definition of accuracy). Figure 3 shows the accuracy of our model as a function of the tolerance where *gmm.tol* ranges between 1.0 and 10^{-8}. As expected, the accuracy increases as the tolerance decreases and flatlines eventually. However, the highest accuracy is not achieved for the smallest tolerance, but for $gmm.tol = 10^{-1}$. A similar trend can be found in the progression of the objective function for GMM which is minimized after 18 iterations as seen in Figure 4. Since we find that a tolerance of 10^{-1} corresponds to 22 iterations, producing the same accuracy as 18 iterations, we set $gmm.tol = 10^{-1}$.

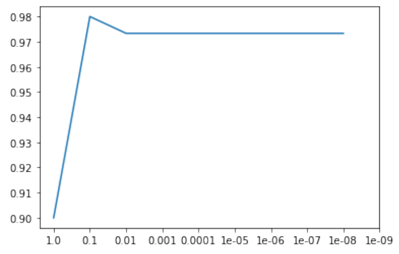

Figure 3: Accuracy as a function of tolerance.

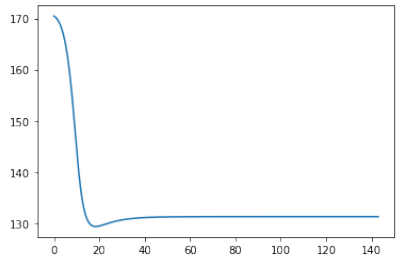

Figure 4: Value of objective function as a function of iterations.

Using this definition of convergence, the final parameters are:

$$\pi_1^f = 0.333, \pi_2^f = 0.306, \pi_3^f = 0.361,$$

$$\mu_1^f = \begin{bmatrix} 2.871 \\ -5.505 \end{bmatrix}, \mu_2^f = \begin{bmatrix} 6.014 \\ -5.089 \end{bmatrix}, \mu_3^f = \begin{bmatrix} 7.532 \\ -5.330 \end{bmatrix},$$

$$\Sigma_1^f = \begin{bmatrix} 0.048 & -0.056 \\ -0.056 & 0.215 \end{bmatrix}, \Sigma_2^f = \begin{bmatrix} 0.362 & -0.218 \\ -0.218 & 0.189 \end{bmatrix},$$
$$\Sigma_3^f = \begin{bmatrix} 0.569 & -0.293 \\ -0.293 & 0.231 \end{bmatrix}.$$

3.3 Analysis and Discussions

Figure 5 shows the cluster assignment provided by the EM algorithm by colouring the data points according to their assigned cluster. The patches represent the elliptical shape of the GMM clusters. As before, the red points represent the cluster centres.

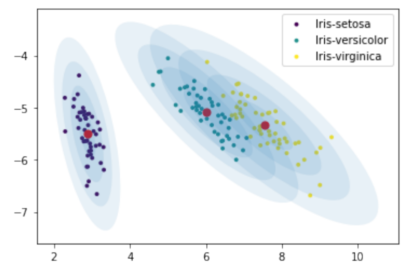

Figure 5: Iris Dataset in 2D with predicted labels (GMM).

A quantitative comparison of the accuracy of *k-means* and GMM is presented in Table 1 below where we define the accuracy score for both models to be the proportion of correctly clustered samples to all samples.

model	k-means	GMM
no. of misclassifications	14	3
accuracy (to 3 dp)	0.887	0.98

Table 1: Accuracy of k-means and GMM compared.

We can see the GMM model quantitatively performs much better than *k-means*, misclassifying only 3 samples instead of 14. The superiority of the GMM model is further illustrated by the fact that after only one iteration of the EM algorithm the GMM model achieves an accuracy of 0.9, already outperforming *k-means*.

Comparing Figures 1 and 2 we realize the misclassifications made by *k-means* mostly occur at the intersection of the clusters corresponding to Iris Versicolor and Iris Virginica while both models classify the samples from the Iris Setosa class perfectly. The GMM model being more capable of distinguishing these neighbouring clusters is not at all surprising since the whole point of implementing GMM was to be able to model non-circular cluster shapes more accurately, accounting for both covariances and means rather than just the means of clusters. Conceptually this is what allows the GMM model to distinguish these classes much better.