Machine Learning - Homework 1

Name: 史天尧　　StudentID：517021910623

1 K-mean algorithm

After initializing the center parameters $\mu_1;\ \mu_2;\ \cdots;\ \mu_K \in R_n$, the K-mean algorithm is to repeat the following two steps until convergence:

1. Assign the points to the nearest μ_i;
2. Update μ_i to be the mean of the data points assigned to it.

Prove that each of the above two steps will never increase the k-mean objective function,

$$J(\mu_K, \cdots, \mu_K) = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}\|x_n - \mu_k\|^2, \tag{1}$$

where

$$r_{nk} = \begin{cases} 1, & \text{if } x_n \text{ is addigned to cluster } k; \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Proof. We give the details of K-means algorithm to prove, that the objective function is non-increasing in every iteration and every stage.

```
Algorithm 1: K-means
Initialize $\mu_i, i = 1, \cdots, K$
repeat
    E-step: for all $x_n \in \chi$ do
        $r_{nk} \leftarrow \begin{cases} 1, & \text{if } k = \arg\min_j \|x_n - \mu_j\|^2 \\ 0, & \text{otherwise.} \end{cases}$
    M-step: for all $\mu_k, k = 1, \cdots, K$ do
        $\mu_k \leftarrow \frac{\sum_n r_{nk} x_n}{\sum r_{nk}}$
until $\mu_i$ converge for all $i$;
```

Let $J(\mu_K, \cdots, \mu_K)^t$ be the value of the objective function immediately after t-th iteration and $J'(\mu_K, \cdots, \mu_K)^t$ be the value of that in $(t+1)$-th iteration after E-step but before M-step. (likewise for r_{nk}) We then show that

$$J(\mu_K, \cdots, \mu_K)^t \geq J'(\mu_K, \cdots, \mu_K)^t \geq J(\mu_K, \cdots, \mu_K)^{t+1}.$$

After E-step, for all x_n, we have

$$\sum_{k=1}^{K} r'_{nk}|x_n - \mu_k\|^2 = \min_j |x_n - \mu_j\|^2,$$

which cannot be greater than $\sum_{k=1}^{K} r_{nk}|x_n - \mu_k\|^2$ as the former result equals some $|x_n - \mu_j\|^2$. Therefore

$$J'(\mu_K, \cdots, \mu_K)^t = \sum_{n=1}^{N}\sum_{k=1}^{K} r'_{nk}|x_n - \mu_k\|^2 \leq \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}|x_n - \mu_k\|^2 = J(\mu_K, \cdots, \mu_K)^t$$

Then let us move to M-step. Compute the partial derivative of μ_k to J and let it equal 0:

$$\begin{aligned}\frac{\partial J}{\partial \mu_k} =& 2\sum_{n=1}^{N} r_{nk}(\mu_k - x_n) = 0,\\ \mu_k =& \frac{\sum_{n=1}^{N} r_{nk}x_n}{\sum_{n=1}^{N} r_{nk}}.\end{aligned}$$

Therefore in M-step, the value we assign to μ_k is exactly the extreme (minimal) point of J. And since J is convex, it becomes the minimum point when r_{nk}s are fixed. From this point, we have $J'(\mu_K, \cdots, \mu_K)^t \geq J(\mu_K, \cdots, \mu_K)^{t+1}$. In conclusion, we have proved in each iteration and step, the objective function is non-increasing. □

2 K-mean vs GMM

Give a variant of k-mean algorithm somewhat between the original k-mean and Expectation-Maximization (EM) for Gaussian Mixture Models (GMM). Please specify the computational details of the formulas. Pseudo-codes of the algorithm would be great.

Discuss the advantages or limitations of your algorithm.

Solution. According to the lecture, this variant can be achieved via multiple approaches. "GMM is more general then K-means by considering mixing weights, covariance matrices, and soft assignments." We can selectively save some feature(s) of GMM and derive a new algorithm. For example, we can modify GMM and let $\pi_i = 1/k, \forall i$ in the initialization stage and fix it, namely preserving all features except mixing weights (prior estimate).

Or we can let the covariance matrix be the identity matrix or multiplied by some constant, or a general diagonal matrix, assigning different weights to different dimensions of data. We can even abort the probabilistic frame but adopt Mahalanobis distance. Nevertheless, here we give an example of letting $\pi_i = 1/k, \forall i$ and fix it.

Algorithm 2: K-mean Variant

```
Initialize $\mu_i, \mathbf{\Sigma}_i, i = 1, \cdots, K$
repeat
    E-step: for all $\mathbf{x}_n \in \chi$ do
        $\gamma(z_{nk}) \leftarrow \frac{\mathcal{N}(\mathbf{x}_n|\mu_k,\mathbf{\Sigma}_k)}{\sum_{j=1}^{K}\mathcal{N}(\mathbf{x}_n|\mu_j,\mathbf{\Sigma}_j)}$
    M-step: for all $\mu_k, \mathbf{\Sigma}_k, k = 1, \cdots, K$ do
        $N_k \leftarrow \sum_{n=1}^{N}\gamma(z_{nk})$
        $\mu_k \leftarrow \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_{nk})\mathbf{x}_n$
        $\mathbf{\Sigma}_k \leftarrow \frac{1}{N_k}\sum_{n=1}^{N}\gamma(z_{nk})(\mathbf{x}_n - \mu_k)(\mathbf{x}_n - \mu_k)^{\mathrm{T}}$
until $\mu_i, \Sigma_i$ converge for all $i$;
```

Advantage of the algorithm includes soft assigning and capturing anisotropy among different dimensions of data. Soft assigning can, to some extent, relive the hungry / dead centroid problem caused by bad initialization, and better describes data points which are in the middle of several clusters: that they do not belong to a certain cluster, but several in between. Capturing anisotropy makes the algorithm better adapted to real-life data, as different dimensions of them usually have different semantics, some of which can be very different in scales.

Disadvantage of this algorithm is also clear: it aborts π_k, thus denying prior knowledge of possible clusters. In practice, domain-specific and expert knowledge is often leveraged as prior to make the performance better.

3 K-mean vs CL

Compare the k-mean algorithm with competitive learning (CL) algorithm. Could you apply the idea of Rival Penalized Competitive Learning (RPCL) to k-mean so that the number of clusters is automatically determined? If so, give the details of your algorithm and then implement it on a three-cluster dataset generated by yourself. If not, state the reasons.

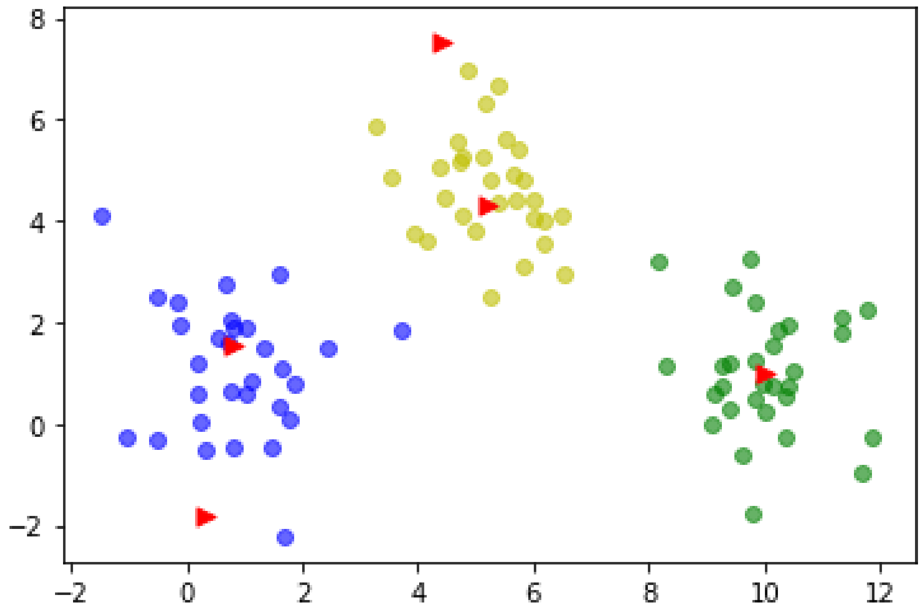
Solution. We have already seen that RPCL can automatically delete redundant clusters when the manually selected k is larger than the actual number of clusters. Each cluster only pulls one μ_i, i.e. the winner near its center, but pushes rivals away, thus determining the number of clusters automatically. However, when the manually selected k is smaller than the actual number of clusters, RPCL does not give an solution to add more clusters. A simple idea would be that to divide the large clusters (i.e. in fact two or more actual clusters, appearing to have large variation) into several smaller clusters, which can be implemented by conducting a $k = 2$ K-means on large clusters, until no cluster is such large.

Therefore, we can conduct RPCL on the dataset at first, delete redundant clusters or split excessively large clusters after converged, and use the result as initialization of the final K-means pass. Here we give the algorithm:

Algorithm 3: RPCL based K-mean

```
Initialize $\mu_i, i = 1, \cdots, K$, max iteration $T$, threshold of number of data points belong to a
  cluster $N_{\min}$ (below which is redundant and to be deleted), threshold of variation of a
  cluster $\sigma_{\max}$ (above which to be split);
repeat
    Conduct RPCL ;
until iteration $< T$ and $\mu_i$ converge for all $i$;
Assign all data points to their nearest $\mu_i$ (E-step);
while no cluster's variation $\geq \sigma_{\max}$ do
    for all clusters do
        if number of data points assigned $< N_{\min}$ then
            delete this cluster;
        if variation $\geq \sigma_{\max}$ then
            conduct $k = 2$ K-mean on this cluster, replace the original center with results
              derived from K-mean;
Use the results $\mu_i, i = 1, \cdots, K'$ as initialization of normal K-mean, conduct K-mean.
```

The experiment (and in Problem 4) is carried out using Python and scikit-learn package. To better visualize the result, we generate a 90 samples, 2-dimensional and 3-cluster dataset from Gaussian distributions. Set $k = 5$, and the result is:



4 Model Selection of GMM

Write a report on experimental comparisons on model selection performance between BIC,AIC and VBEM.

Specifically, you need to randomly generate datasets based on GMM, by varying some factors, e.g., sample sizes, dimensionality, number of clusters, and so on.

1. BIC, AIC: First, run EM algorithm on each dataset X for $k = 1, \cdots, K$ and calculate the log-likelihood value $\ln[p(X|\hat{\Theta}_k)]$ is the maximum likelihood estimate for parameters; Second, select the optimal k^* by

$$k^* = \arg\max_{k=1,\cdots,K} J(k), \tag{3}$$

$$J_{AIC}(k) = \ln[p(X|\hat{\Theta}_k)] - d_m, \tag{4}$$

$$J_{BIC}(k) = \ln[p(X|\hat{\Theta}_k)] - \frac{\ln N}{2} d_m. \tag{5}$$

2. Use VBEM algorithm for GMM to select the optimal k^* automatically or via evaluating the lower bound.

Solution. We carry out the comparisons following the sequence of different datasets:

1. **Dimensionality=2, Sample size=500 for each cluster, Number of clusters=2**

- **AIC**

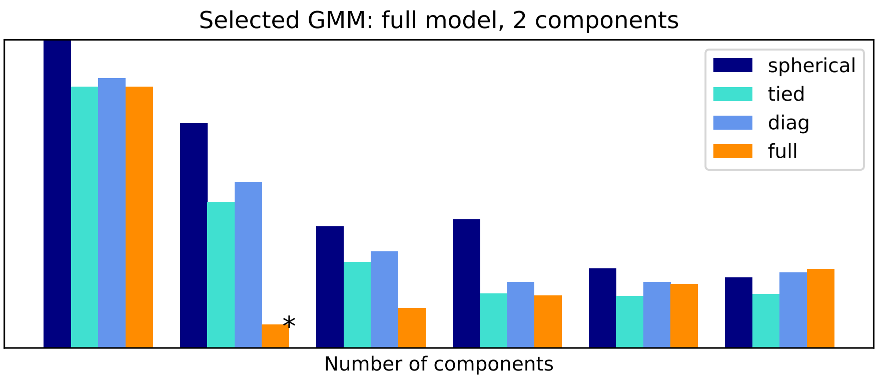


图 1: AIC for choosing k^* in dataset 1

- **BIC**

  AIC and BIC creterion give the same GMM model in this dataset. Below is the visualization. However, we can see that BIC is less likely to choose wrong k^*.

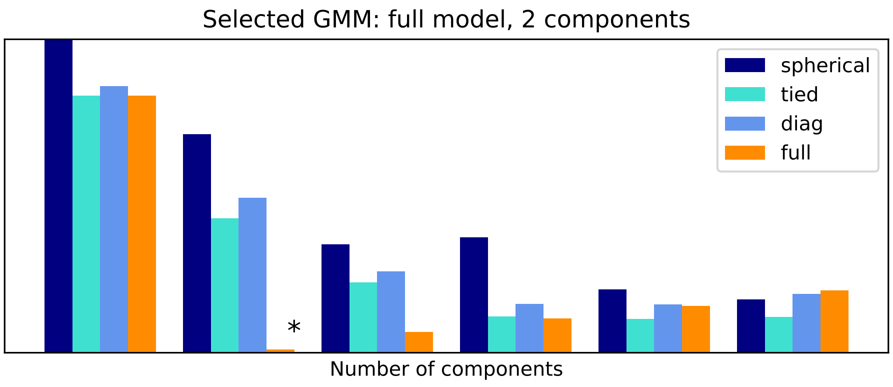


图 2: BIC for choosing k^* in dataset 1

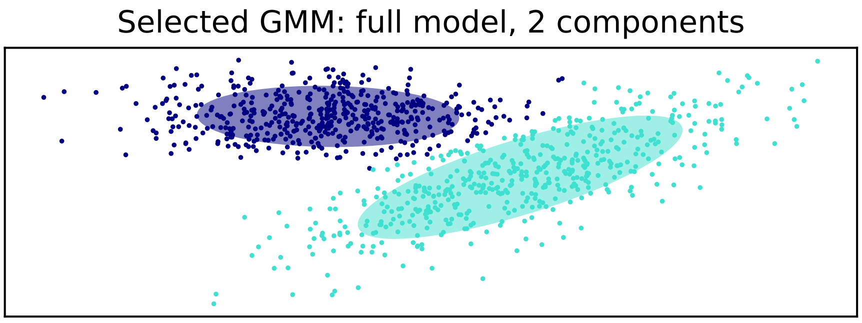


图 3: The chosen GMM in dataset 1

- **VBEM**
  As implied in the VBEM algorithm, take a conservative stand, and fit only the most possible components. However, when testing on this dataset, it goes into problems. As shown below, it only partly works when selecting $k = 5$, and VBEM can tell there is one redundant cluster. But for $k = 3, 4$, it is not satisfying.

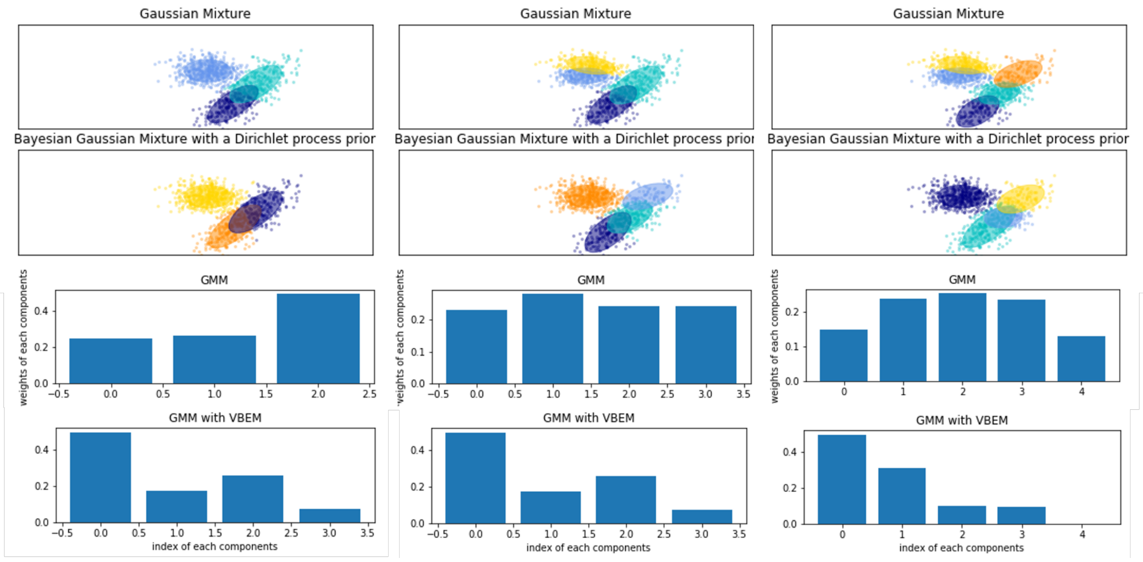


图 4: VBEM for $k = 3, 4, 5$ in dataset 1

Why does this happen? I try to modify the dataset and pull the two cluster away. This time, the VBEM works. This implies that when the clusters are actually close to each other, VBEM may fail to tell out the redundant clusters. Or in other words, there is not enough evidence for VBEM to reduce clusters, as the number provided seems just fine.

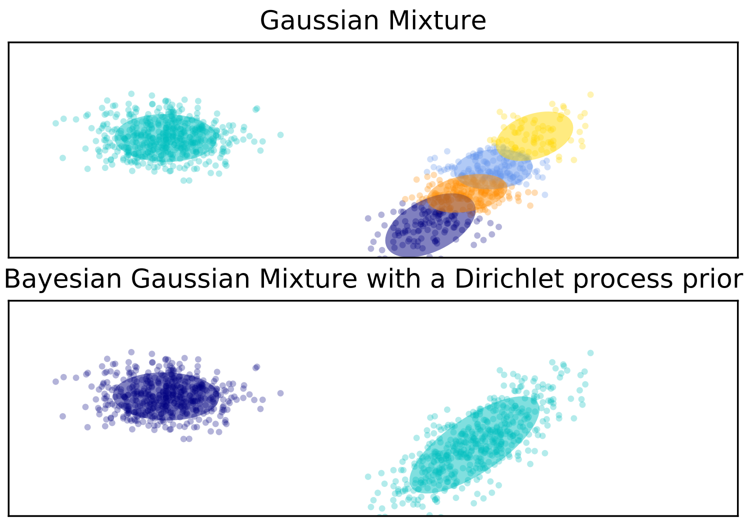


图 5: VBEM vs GMM in modified dataset 1

2. **Dimensionality=2, Sample size=1000 for each cluster, Number of clusters=2**

- **AIC**

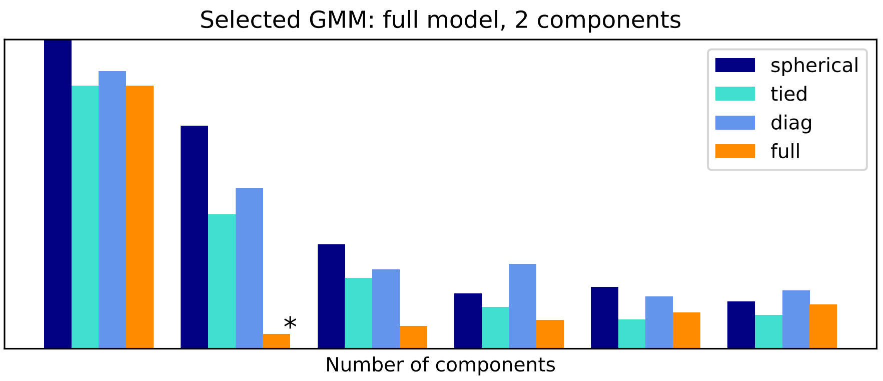


图 6: AIC for choosing k^* in dataset 2

- **BIC**

  The pattern is the same as in dataset 1.

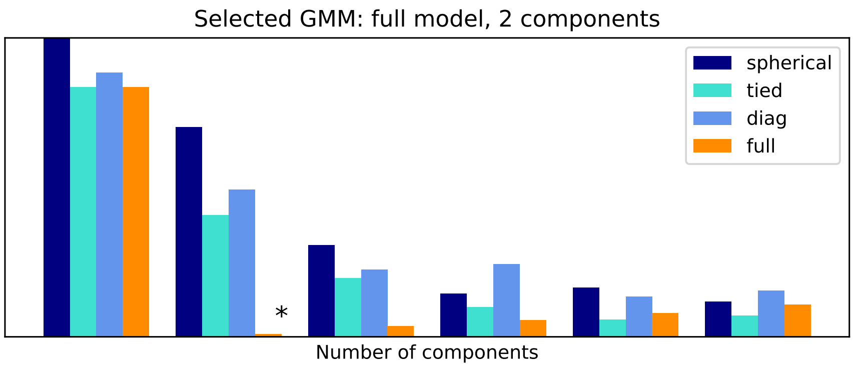


图 7: BIC for choosing k^* in dataset 2

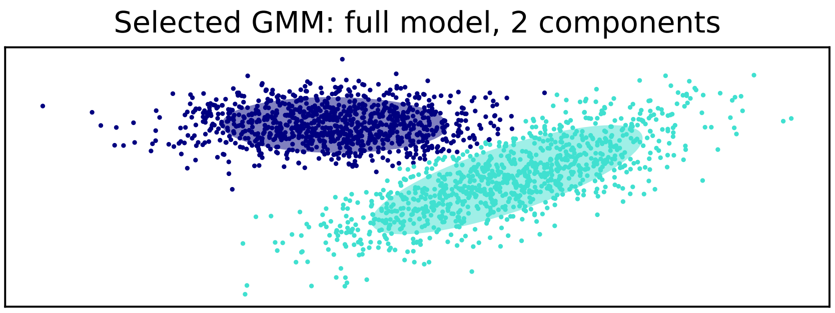


图 8: The chosen GMM in dataset 2

- **VBEM**
  The same problem happens, or even worsens this time—as the sample size increases, there are more outliers and harder for VBEM to tell. Even though the two clusters are pulled away, VBEM still fits one redundant cluster.

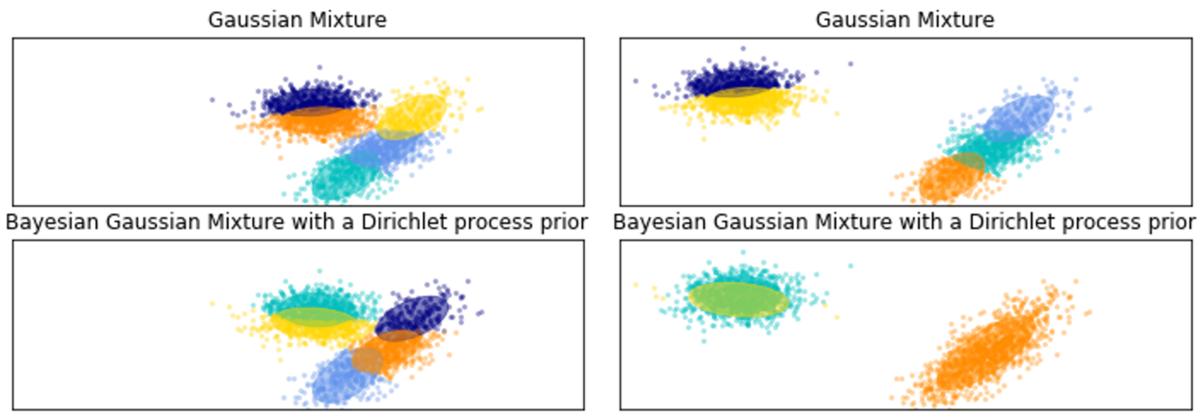


图 9: VBEM vs GMM in original and modified Dataset 2

3. **Dimensionality=2, Sample size=1000 for each cluster, Number of clusters=3**

- **AIC**

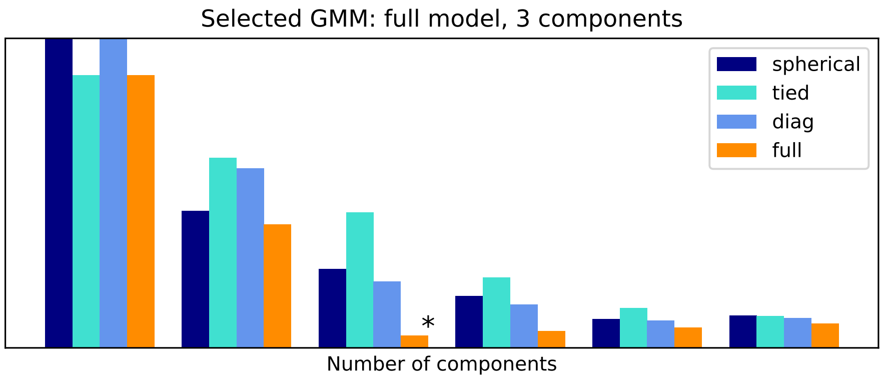


图 10: AIC for choosing k^* in dataset 3

- **BIC**

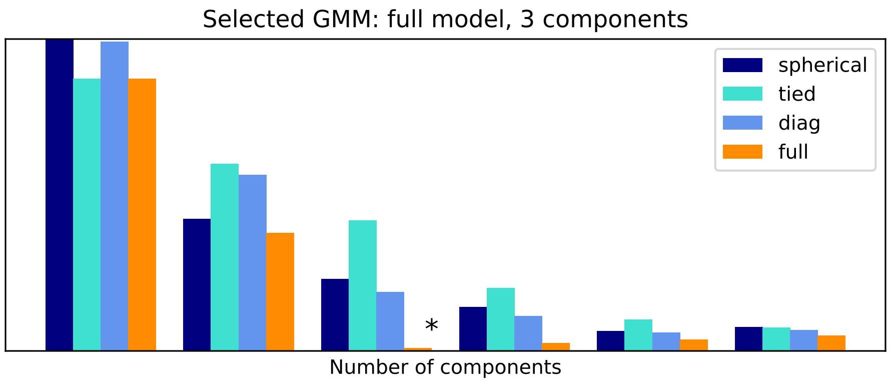


图 11: BIC for choosing k^* in dataset 3

The pattern is the same as in dataset 1.

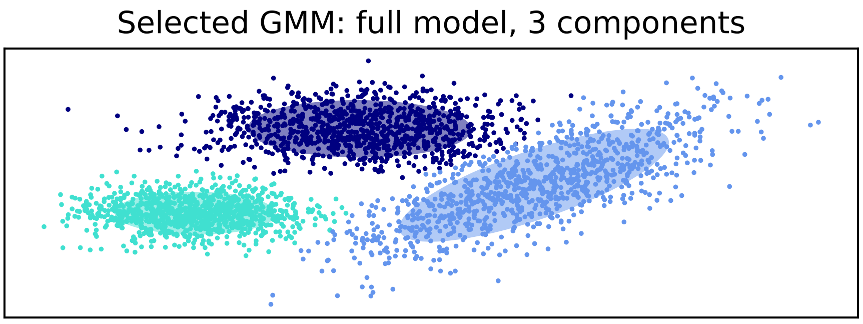


图 12: The chosen GMM in dataset 3

- **VBEM**

  The pattern is the same as in dataset 1. Better than in dataset 2, maybe because of the decreased $k/\#cluster$ ratio. This implies that determining k automatically is really tricky and relies on setting of some key parameters.

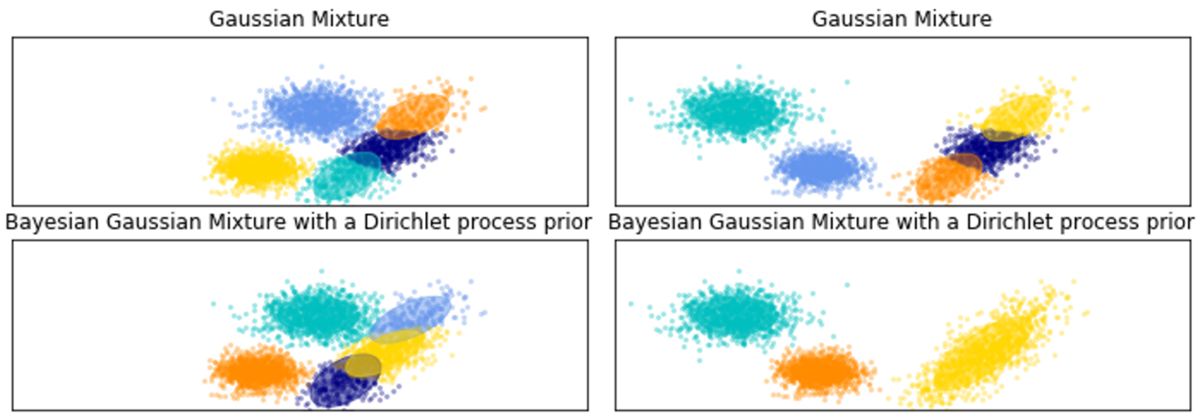


图 13: VBEM vs GMM in original and modified Dataset 3

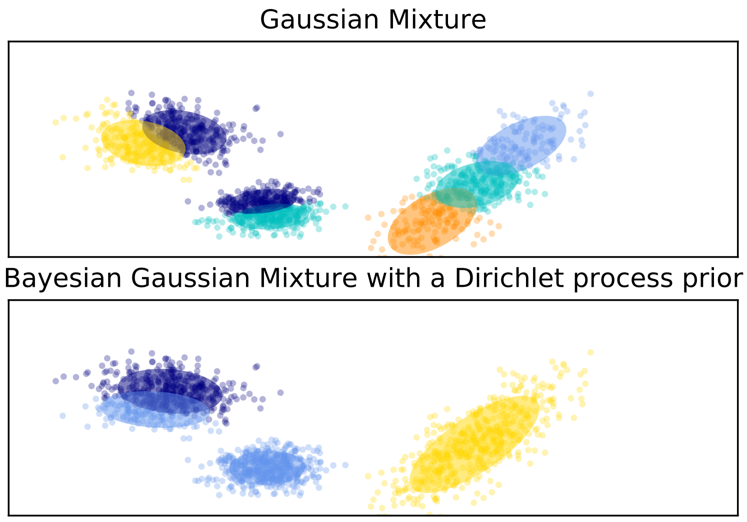


图 14: VBEM fails again when k set to 7

4. **Dimensionality=10, Sample size=500 for each cluster, Number of clusters=2**

   - **AIC**
   - **BIC**

     10-dimensional data cannot be visualized, though AIC and BIC still agree with each other. A major change is that spherical model is selected, which agrees with the way I generated 10-dim data—for convenience I did not apply transformation to the data drawn from the standard Gaussian distribution.

   - **VBEM**

     This time VBEM performs fairly—better than low dimensional results, though unable

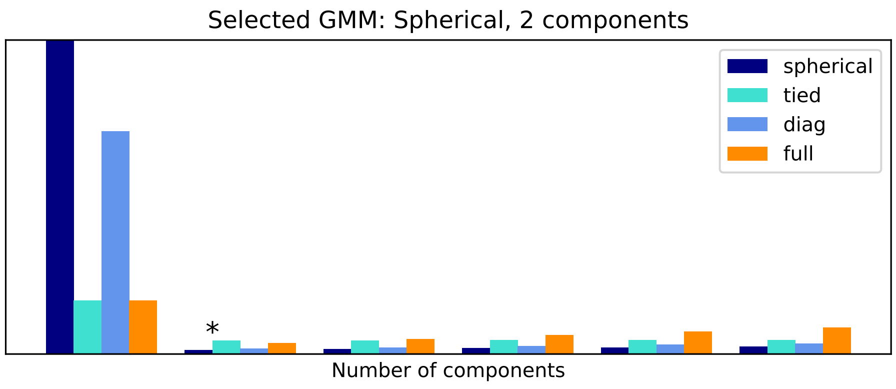


图 15: AIC for choosing k^* in dataset 4

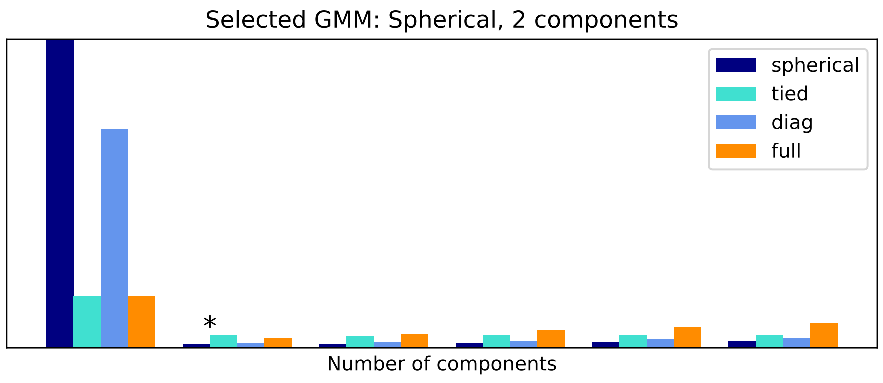


图 16: BIC for choosing k^* in dataset 4

to reduce all redundant clusters, but keeps all redundant clusters very small (weight of cluster 2 is not 0).

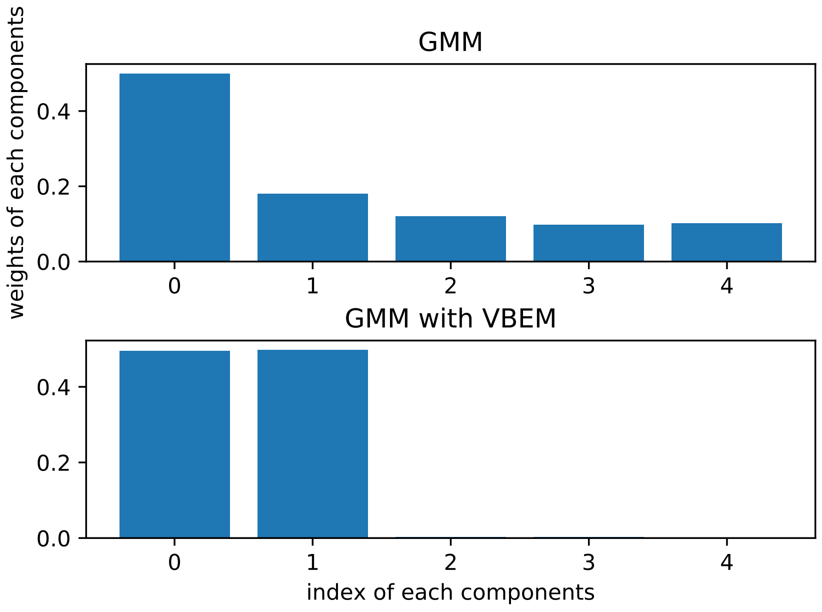


图 17: VBEM for choosing k^* in dataset 4