# Report on Homework 1

CS420, Machine Learning, Shikui Tu, Summer 2018

## 1 k-means vs GMM

Basically, k-means employs One-in-K assignment while GMM utilizes soft assignment. It turns out that k-means has bad robust under some circumstances. Therefore, I tend to introduce some of the ideas in GMM into k-means to form a new variant of k-means.

There are two main differences in my variant of k-means algorithm compared with the original:

- I introduce parts of the soft assignment into k-means to make it more robust.
- I affiliate a hyper parameter $Thres$ into original k-means. When the posteriori probablity is larger than $Thres$, it can be retained, or it would be 0.

The details of my algorithm can refer to the pseudo-codes in Alg. 1 (The meanings of most symbols are same as the ones in GMM).

**Algorithm 1:** A variant of k-means

**Input :** The number of clusters $K$

**Output:** $\pi_k, \mu_k, \Sigma_k, (k = 1, 2, ..., K)$

Initialize the means $\mu_k$, covariances $\Sigma_k$, mixing coefficients $\pi_k$ and threshold $Thres$;

Evaluating the initial value of the log likelihood;

**while** *the convergence criterion of parameters or log likelihood is not satisfied* **do**

**E step.** Evaluate the responsibilities with the current parameter values:

$$\omega \leftarrow \frac{\pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(x_n|\mu_j, \Sigma_j)} \quad , \quad \gamma(z_{nk}) \leftarrow \begin{cases} \omega & \omega > Thres \\ 0 & \omega \leq Thres \end{cases}$$

$$z_n \leftarrow \frac{e^{z_{n_i}}}{\sum_{j=1}^{K} e^{z_{n_j}}}$$

**M step.** Re-estimate the parameters with the current responsibilities:

$$N_k \leftarrow \sum_{n=1}^{N} \gamma(z_{nk}) \quad , \quad \mu_k^{new} \leftarrow \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) x_n$$

$$\Sigma_k^{new} \leftarrow \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(x_n - \mu_k^{new})(x_n - \mu_k^{new})^T$$

$$\pi_k^{new} \leftarrow \frac{N_k}{N}$$

Evaluate the log likelihood:

$$\ln p(X|\mu, \Sigma, \pi) \leftarrow \sum_{k=1}^{K} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(x_n|\mu_k, \Sigma_k) \right\}.$$

**return** $\pi_k, \mu_k, \Sigma_k, (k = 1, 2, ..., K)$;

With the introduction of probability thoughts, my algorithm is somewhat similar to EM algorithm and would be more robust than k-means. Besides, the $Thres$ would eliminate the negligible posteriori probabilities and improve accuracy under the premise of robustness.

Generally speaking, more hyper parameters would result in more tricks, my algorithm is no exception. The choice of $Thres$ may greatly affect the result.

Additionally, my algorithm also encounters the initialization and unknown $K$ problems. But I think such problems do not contradict with my core ideas, and could be resolved by introducing more components (e.g. the idea of RPCL), which would be discussed in Sec. 2.2

# 2 k-means vs CL

## 2.1 Comparison between k-means and CL

- **Similarities**

  1. They can not estimate the total number of clusters just from data.
  2. The effects of these two algorithms highly rely on initialization.
  3. Both of them have huge potential for expansion to improve their robustness.
  4. They consider data in distance framework instead of probability.

- **Differences**

  1. k-means is a batch learning algorithm while CL belongs to adaptive learning.
  2. CL has a hyper parameter $\eta$ to adjust, making the algorithm a little tricky. But k-means has no hyper parameter.
  3. k-means would take more time to converge when it comes to large data size. CL avoids this issue by updating the centers by one data point each time.
  4. Comparing to k-means, CL requires less computing resource (CPU, memory and etc).
  5. The process of k-means is more simple and succinct than CL.

## 2.2 Application of RPCL in k-means

Since k-means can not estimate the total number of clusters from data while RPCL can make extra centers far away to control the number of clusters, I tend to apply the idea of RPCL to **the initialization of k-means**.

Suppose the initial number of centers is $m$ and the corresponding weight vectors are $v_i(i = 1, ..., m)$. For each center, its output could be defined as:

$$u_i \leftarrow \begin{cases} 1 & \text{if } u_i \text{ is the winner} \\ -1 & \text{if } u_i \text{ is the rival} \\ 0 & \text{otherwise} \end{cases}, \tag{1}$$

To make the winner approach the target cluster faster and keep the rival away from it more efficiently, I take the geometric distribution of the data points into consideration. A density function for each data point is hence defined as:

$$\rho(x_j) = \exp[-\sum_{i=1}^{N} \frac{d(x_i, x_j)}{\sum_{i=1}^{N} d(x_1, x_j)}], j = 1, 2, ..., N \tag{2}$$

where $d$ denotes the distance function (e.g. Euclidean distance).

Then the adjustment for each weight vector $v_i$ can be conducted according to $u_i$ and $\rho(x_j)$:

$$\Delta v_i = \begin{cases} \alpha\rho(x_j)(x_j - v_i) & u_i = 1 \\ \beta\rho(x_j)(x_j - v_i) & u_i = -1 \\ 0 & \text{otherwise} \end{cases}, \tag{3}$$

where $\alpha$ and $\beta$ are hyper parameters.

Once the weight vectors converge, I will eliminate negligible ones according to a threshold $T$. Then the number of the remaining centers should be the clusters number - $K$. Besides, these centers would locate in 'good' initial positions.

The pseudo-codes of my algorithm can refer to Alg. 2.

**Algorithm 2:** The idea of RPCL in k-means

**Input :** The dataset

**Output:** The initial $m$ centers and corresponding weight vectors $v_i(i = 1, ..., m)$

Initialize the number of centers $m$ and weight vectors $v_i$, threshold $T$, density $\rho(x_j)j = 1, 2, ..., N$.

**while** *the convergence criterion is not satisfied* **do**

  **for** *i = 0 to n* **do**

    Calculating $u_i$ according to Eqn. 1.

    Adjust the weight vectors of each center via Eqn. 3.

Assign each data point to its closest center.

**for** *i = 0 to m* **do**

$$\eta \leftarrow \frac{N_i}{N},$$

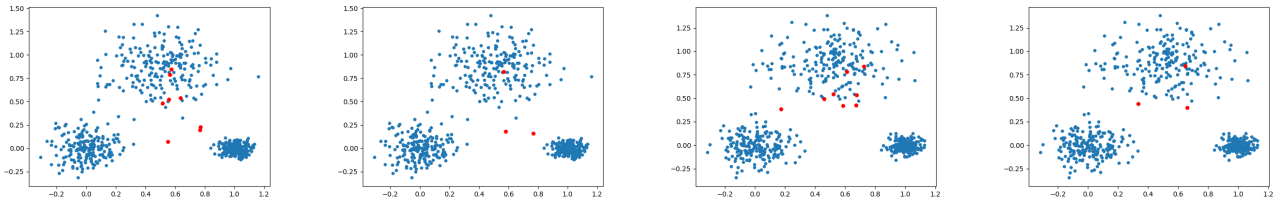
  where $N_i$ denotes the number of data points that belong to $ith$ center, $N$ is the number of data points in the dataset.

  **if** $\eta < T$ **then**

    Remove center $v_i$.

**return** $v$;

I test this application in a three-cluster dataset generated by myself, and the results of initial centers choice can refer to Fig. 1.



(a) Initialization_1 (b) Result_1 (c) Initialization_2 (d) Result_2

Figure 1: The results of RPCL in kmeans. Blue points denotes data and red points denotes cluster centers.

It turns out that the idea of RPCL could determine the number of clusters and even provides a good initialization, which might greatly speed up the process of kmeans.

# 3 Model Selection of GMM

In this section, I compare model selection performances among BIC, AIC and VBEM by varying the sample sizes and the cluster center distances of the dataset (All the datasets are randomly generated based on GMM).

## 3.1 Different Sample Sizes

In this experiment, I genetate 4 clusters on the four corners of a square, and run BIC, AIC and VBEM with different sample sizes. Then I visualize the results to check their performaces, which are shown in Fig. 2. The corresponding line chart can refer to Fig. 3.

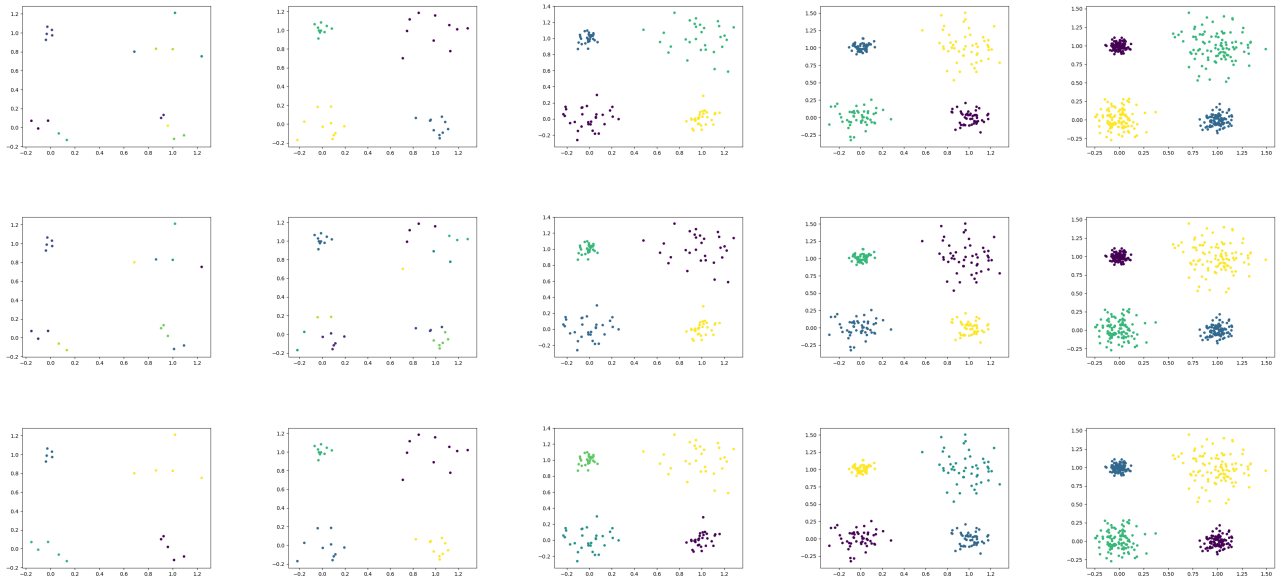


Figure 2: The model selection performances of BIC, AIC and VBEM on different sample sizes. Each row from up to bottom denotes BIC, AIC and VBEM respectively. Each column from left to right denotes different sample sizes (5, 10, 30, 50, 100).

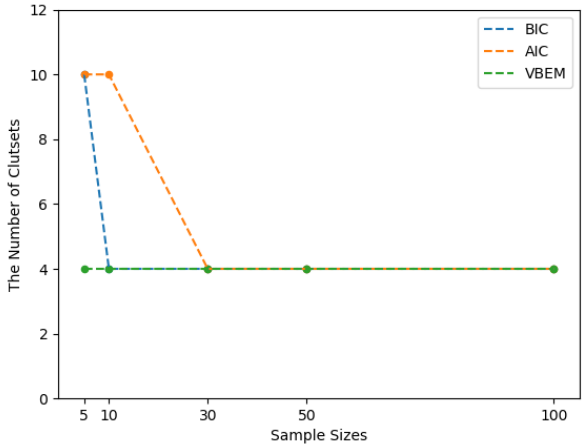



Figure 3: The Model Selection Performances of BIC, AIC and VBEM on Different Sample Sizes.

It can be found from the results that all of the three approaches perform well in middle or large sample

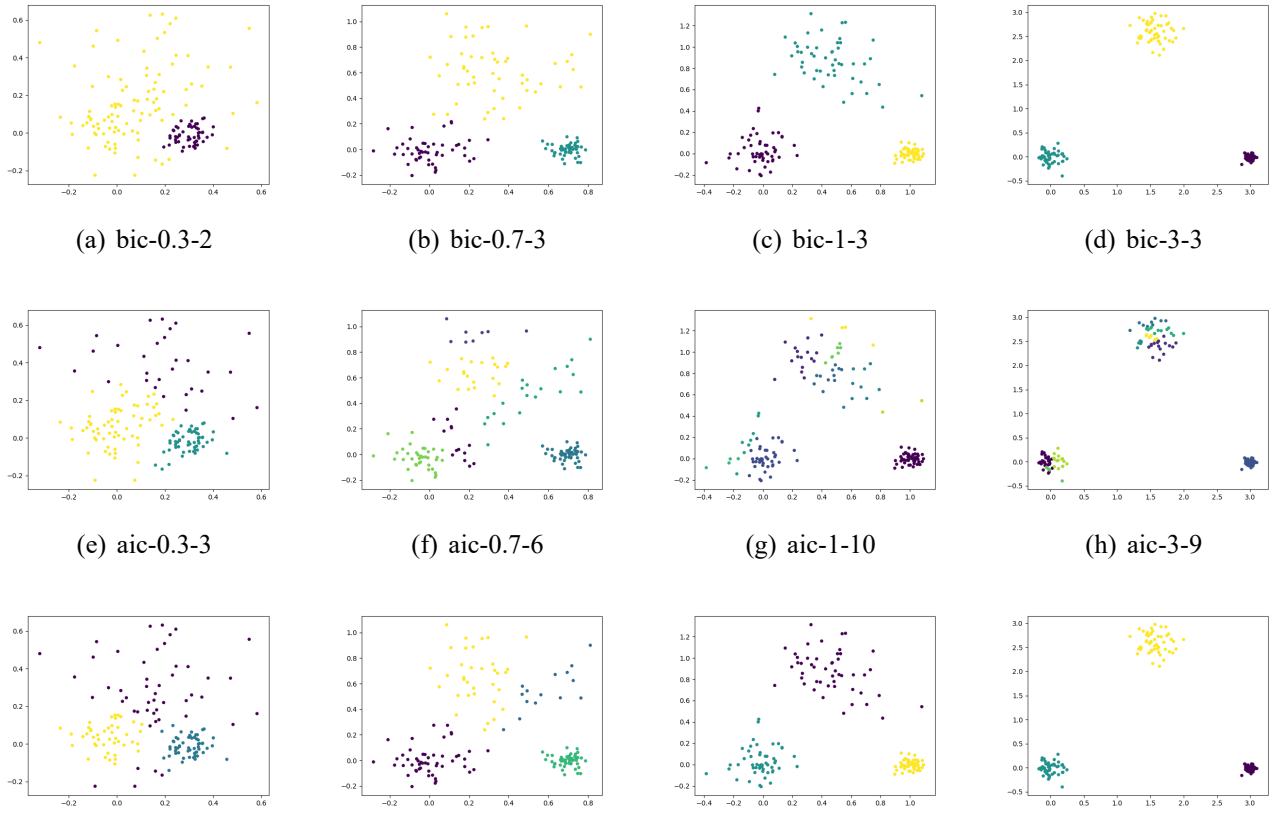
sizes. When it comes to small sample sizes, BIC and AIC are more likely to get poor performance while VBEM can prevent such cases. More exactly, BIC outperforms AIC slightly in small sample sizes. I infer that this is because BIC has taken sample sizes into account while AIC has not.

Anyhow, in this experiment, VBEM achieves best clutsering result, BIC is the next and AIC is the worst.

## 3.2 Different Distance Among Cluster Centers

After exploring the effects of sample sizes on the three approaches, I feel curious about how the distance among cluster centers affects their performances.

I generate 3 clusters on the three corners of a equilateral triangle, increasing the distance among them and visualizing the model selection results in Fig. 4.



(a) bic-0.3-2 (b) bic-0.7-3 (c) bic-1-3 (d) bic-3-3

(e) aic-0.3-3 (f) aic-0.7-6 (g) aic-1-10 (h) aic-3-9

(i) vbem-0.3-3 (j) vbem-0.7-4 (k) vbem-1-3 (l) vbem-3-3

Figure 4: The model selection performances of BIC, AIC and VBEM on different cluster center distances. Each row from up to bottom denotes BIC, AIC and VBEM respectively. The two numbers below each figure represent the distance among three cluster centers and the $K$ chosen by corresponding approach respectively.

As is shown in the results, BIC and VBEM outperform AIC a lot, especially in long distance cases, where AIC tends to choose models having more cluster centers while BIC and VBEM could achieve appropriate model selection.

Combined with the results in Sec. 3.1, another significant phenomenon could be discovered. All the three approaches will obtain favorable performance under dataset with high cluster density. Nevertheless, AIC is prone to collapse when facing low cluster density, where BIC and VBEM are more applicable.

## 3.3 Discussion & Conclusion

In these experiments, I mainly compare the model selection ability of BIC, AIC and VBEM under different sample sizes and cluster center distances. From the results of Sec. 3.1 and Sec. 3.2, VBEM performs best, BIC is just a little bit worse and AIC performs worst.

Actually, though these experiments can reveal their performance to a certain extent, they are only a small portion of comparisons among the three approaches and the results may be variant in other conditions. To be more accurate and comprehensive, more kinds of contrast experiments are needed (e.g. different data dimensions, number of clusters).

Ultimately, I want to express my sincere thanks to Prof. Shikui Tu and TA Yajing Chen for their patient guidance and great help in this homework! Thank you!