Machine Learning Homework 1

Sun Hao
Student ID: 516030910362

1 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

The original k-mean has a simple step. It circulates to find a central point in each cluster and re-cluster each point in data set until convergence. In the detailed realization, each point is assigned into its closest cluster by calculating the euclidean distance away from cluster center, which is called **hard-assignment**. I'll give a variant of **soft-assignment** k-mean algorithm, inspirated from the realization in Expectation Maximization(EM) for Gaussian Mixture Models(GMM).

1.1 Computational detial of the formulas

Recall that in original(hard-assignment) k-mean, the object function can be formally defined as:

$$\text{minimize } J = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk} \left\|x_n - \mu_k\right\|^2$$

where N is the number of samples(points), K is the number of cluster. μ_k is the central point of cluster k, and $r_{nk} = 1$ if and only if data point x_n is assigned to cluster k; otherwise $r_{nk} = 0$.

In the soft-assignment k-mean, the object function is totally same as the original one.

However, The original K-means algorithm uses hard assignment mechanism and each data point belongs to its closest cluster center. EM algorithm, on the other hand, uses soft assignment mechanism and each data point is assigned to every cluster center according its probability of generating the data. In practice EM can give more accurate results than K-mean specially for GMM, but this comes at the expense of higher computational cost. In order to simplify EM computation process, the following soft assignment K-means algorithm. The data point x_n can be assigned to **every** cluster, and the membership value associated with each cluster is as follows:

$$w_n\left(C_k\right) = \frac{e^{-\lambda\left\|x_n-\mu_k\right\|^2}}{\sum_{k=1}^{K} e^{-\lambda\left\|x_n-\mu_k\right\|^2}} \tag{1}$$

where it is obviously satisfied that $\sum_{k=1}^{K} w_i(C_k) = 1$ for all $1 \le n \le N$. By doing this, we can separate a point into k parts with different "weights". Then, similar as the original one, The central point of k^{th} cluster μ_k is calculated by

$$\mu_k = \frac{1}{N_k}\sum_{n=1}^{N} w_n\left(C_k\right) x_n \tag{2}$$

where we define N_k as

$$N_k = \sum_{n=1}^{N} w_n\left(C_k\right) \tag{3}$$

Note that N_k may not be a integer, but that doesn't matter at all.

When μ_k for all $1 \le k \le K$ has been stable or converged, we have to assign every point to a certain cluster. Therefore, in the final step, we assign points according to the original method, that is, assigning every point to its closest cluster.

1.2 Pseudo-code

Input: the point data $\{x_n\}$, the number of clusters K
Output: the assignment array $r[N][K]$, the definition of $r[n][k]$ is in section 1.1

Initialize $\mu[k]$ by selecting some random initial values for $\mu[k]$, $1 \leq k \leq K$;
//larger distance between $\mu[k]$ is better;
Initialize $\mu'[k]$, $1 \leq k \leq K$;
while $\exists k, \mu'[k] \neq \mu[k]$ **do**
    $\mu'[k] \leftarrow \mu[k]$ for $1 \leq k \leq K$;
    for *each data point* $x[n]$ **do**
        Calculate $w_n(C_k)$ of each cluster k according to equation 1;
        Compute new cluster centers $\mu[k]$ according to equation 2 6;
    end
end
for *each data point* $x[n]$ **do**
    $k \leftarrow \underset{1 \leq k \leq K}{\operatorname{argmax}}(w_n(C_k))$;
    $r[n][k] \leftarrow 1$;
    $r[n][j] \leftarrow 0$ if $j \neq k$;
end
return $r[N][K]$;

Algorithm 1: soft-assginment k-mean

1.3 Advantages and limitations

Advantages

- Compared to original k-mean, the soft-assignment k-mean has a more accurate result. Because as we know, It is more powerful to consider everything in probability framework, and the weights $w_n(C_k)$ is exactly describing probability(likely).
- Compared to EM algorithm, this method doesn't have to calculate the covariance matrix and therefore the speed improves a lot.

Limitations

- It still has the most of the problems which exist in original k-means, such as hardness of determining K, still using euclidean distance.
- It performs not well when points data uniforms Gaussian Mixture Distribution.

2 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

2.1 Comparison between k-mean and CL

Recall CL is usually used for making data clustering by a number of units or agents. Each agent is featured by a parametric point $_j = m_j$ that moves among the observation samples and seeks to be located at the center of a cluster of samples, such that several points collectively perform clustering analysis. As a sample x_t comes, each m_j evaluates a degree of its suitability on representing x_t

by an individual value or criterion, e.g., $\varepsilon_t(\theta_j) = \|x_t - m_j\|^2$, and competes to be allocated to represent x_t . The competition is guided globally by the winner-take-all policy as follows:

$$p_{j,t} = \begin{cases} 1, & \text{if } j = c, \\ 0, & \text{otherwise;} \end{cases} \qquad c = \arg\min_j \varepsilon_t(\theta_j) \tag{4}$$

Then, each m_j is updated to further reduce $\varepsilon_t(\theta_j)$ by

$$m_j^{new} = m_j^{old} + \eta p_{j,t}\left(x_t - m_j^{old}\right) \tag{5}$$

Similarities

- One has to choose k first when using whether K-mean or CL.
- K-mean and CL are both sensitive to the initialization of the cluster centers m_k.
- K-mean and CL are both measured in Euclidean space.

Differences

- Because CL assigns data points one by one, CL is very sensitive to the order of data point sequence while k-mean result is absolutely stable over the order change of data point sequence.
- CL needs only one loop for each points, however k-mean repeat the progress: re-choosing cluster center points, assigning each points, until all cluster centers become unchanged. k-mean has a higher time complexity.
- Compared to k-mean, CL is more likely to get trapped in the local optimal solution because of the sensitivity to order of data points sequence. Thus, we can say k-mean has a better effect.

2.2 Improved k-mean based on RPCL

The Rival Penalized Competitive Learning(RPCL) differs from CL in that

$$\varepsilon_t(\theta_j) = \alpha_j \|x_t - m_j\|^2 \tag{6}$$

$$p_{j,t} = \begin{cases} 1, & \text{if } j = c, \\ -\gamma, & \text{if } j = r, \\ 0, & \text{otherwise} \end{cases} \quad \begin{cases} c = \arg\min_j \varepsilon_t(\theta_j) \\ r = \arg\min_{j \neq c} \varepsilon_t(\theta_j) \end{cases} \tag{7}$$

where α_j is the frequency that the j^{th} agent won in past, and γ approximately takes a number between 0.05 and 0.1 for controlling the penalizing strength.

By adding this rival penalized mechanism makes extra agents driven far away from data as long as the number of agents is initially given to be larger than the number of clusters to be detected. In other words, RPCL can allocate an appropriate number of learning agents to detect a correct number of clusters or objects automatically during learning. This automatic allocation is a favorable nature that both CL and FSCL as well as the conventional clustering algorithms (e.g., k-means) do not have.

Thus, we can use RPCL to get the most appropriate number k of clusters and the k initial cluster centers in k-mean. By doing this, I combine both ways and it cost not much extra time because k-mean gets convergence fast from a very good cluster centers initialization.

2.2.1 Pseudo-code

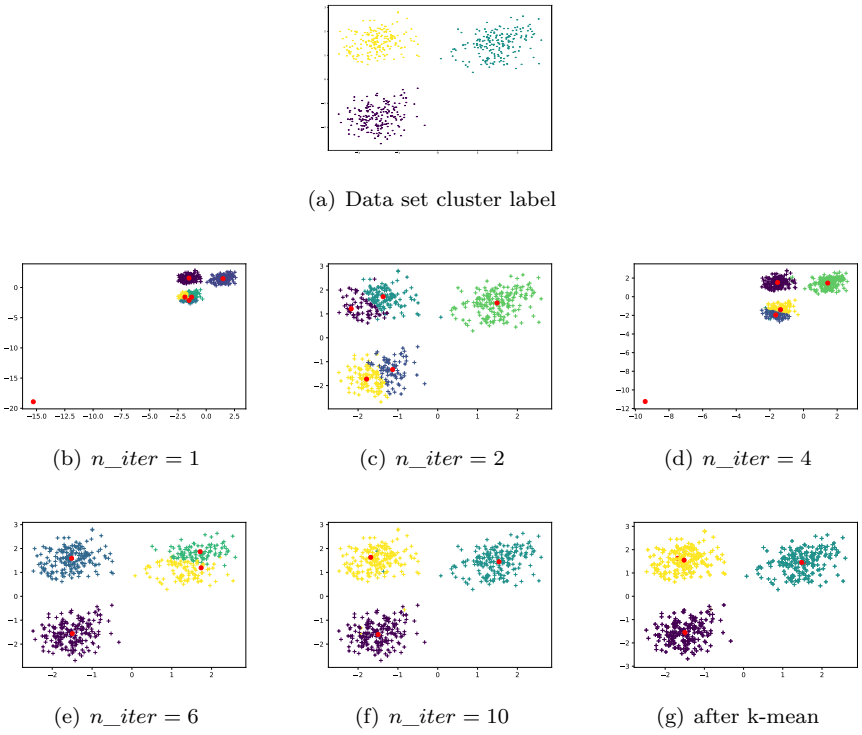
2.2.2 Implementation

For visualization, the data point is 2 dimension. I generated 600 points and both x axis range and y axis range of all points are from -3 to 3. The data set cluster label value is shown as Figure 1(a).

In the algorithm, I set the parameters as, rival penalized coefficient $\gamma = 0.05$, max iterations $n\_iter = 10$, removing cluster ratio threshold $\lambda = \frac{1}{3}$, and the initial number of clusters $init\_k = 6$. The whole algorithm progress is shown as Figure 1.

Input: the point data $\{x_n\}$
Output: the assignment array $r[N][K]$, the definition of $r[n][k]$ is in section 1.1
Parameters initialization: rival penalized coefficient γ, max iterations n_iter, ratio threshold λ to remove useless cluster centers, and the initial number of clusters $init\_k$;
//confirm $N \geq K$, K is the optimal number of clusters;
$k \leftarrow init\_k$;
shuffle($\{x_n\}$);
$centers[k] = \text{sample}(\{x_n\}, k)$;
for $i \leftarrow 0$, *to* n_iter **do**
    shuffle($\{x_n\}$);
    Initial array $n\_point\_of\_each\_cluster[t] = 0$ for $1 \leq t \leq k$;
    for $n \leftarrow 1$ *to* N **do**
        Calculate the penalized distances away from each cluster center of point x_n according to Equation 6.;
        Update array $centers$ according to Equation 5, 7;
        Update array $n\_point\_of\_each\_cluster$;
    end
    for $t \leftarrow 1$ *to* k **do**
        $remove\_count \leftarrow 0$;
        if $n\_point\_of\_each\_cluster[t] < \lambda \cdot \frac{N}{k}$ **then**
            Remove cluster t;
            $remove\_count \leftarrow remove\_count + 1$
        end
        $k \leftarrow k - remove\_count$;
    end
end
Run k-mean algorithm with k, initial center position array $centers$;

Algorithm 2: Improved k-mean based on RPCL



(a) Data set cluster label

(b) $n\_iter = 1$ (c) $n\_iter = 2$ (d) $n\_iter = 4$

(e) $n\_iter = 6$ (f) $n\_iter = 10$ (g) after k-mean

Figure 1: The program running progress of Algorithm 2

2.2.3 Another method

The algorithm 2 makes RPCL and k-mean separated(first execute RPCL and then execute k-mean). Here comes another method to merge the two algorithm together.

As we all know, RPCL automatically select the best k by pushing away unnecessary center points. Similarly, in k-mean, we can still put this idea into it.

In loops in k-mean, recall that each time after assigning all points, we need to re-calculate the center points. However, in improved k-mean, instead of getting the center point, we calculate the m_k by Equation 5, 6 and 7, which makes push away unnecessary center points.

3 Model selection of GMM

3.1 Generating date set based on GMM

The generating code is shown as below.

```
import numpy as np
import matplotlib.pyplot as plt
n_component=np.random.randint(2,11)
n_samples = np.random.randint(40,200,size=n_component)
n_dimension=np.random.randint(2,51)
random_mu=100*np.random.random((n_component,n_dimension))
random_sigma=10*np.random.random((n_component,n_dimension))

X=np.zeros((0,n_dimension))
label_y=np.zeros(0)
for component_index in range(n_component):
    X=np.concatenate((
            X,random_sigma[component_index]*np.random.randn(
                    n_samples[component_index],n_dimension)+\
                    random_mu[component_index]),axis=0)
    label_y=np.concatenate((
            label_y,component_index*np.ones(
                    n_samples[component_index])
            ))

# only show 2 dimension
plt.scatter(X[:,0],X[:,1],c=label_y,s=3)
```

I tried to make the generated dataset based on GMM contain as many random parameter variables as possible to test more cases. Here is the table to show all variables' range.

variable name	description	range
n_component	number of components(clusters) in GMM	$2 \sim 10$
n_samples	number of samples of each component	$40 \sim 100$
n_dimension	number of dimensions of each sample	$2 \sim 50$
random_mu	μ in normal distribution $\mathbf{N}(\mu, \sigma)$ of each component	$0 \sim 100$
random_sigma	σ in normal distribution $\mathbf{N}(\mu, \sigma)$ of each component	$0 \sim 10$

Table 1: Parameters in generating dataset based on GMM

3.2 BIC and AIC

Recall BIC(Bayesian Information Criterion) and AIC(Akaike's Information Criterion) can be calculated by the formula

$$k^* = \arg \max_{k=1,\ldots,K} J(k) \tag{8}$$

$$J_{AIC}(k) = \ln \left[p\left(X|\hat{\Theta}_k\right) \right] - d_m \tag{9}$$

$$J_{BIC}(k) = \ln\left[p\left(X|\hat{\Theta}_k\right)\right] - \frac{\ln N}{2} d_m \tag{10}$$

The time complexity of BIC and AIC is almost the same because the formulas are similar. Therefore, in the performance comparison between AIC and BIC, I tend to focuse more on whether the output k_{AIC} and k_{BIC} is equal to k^*.

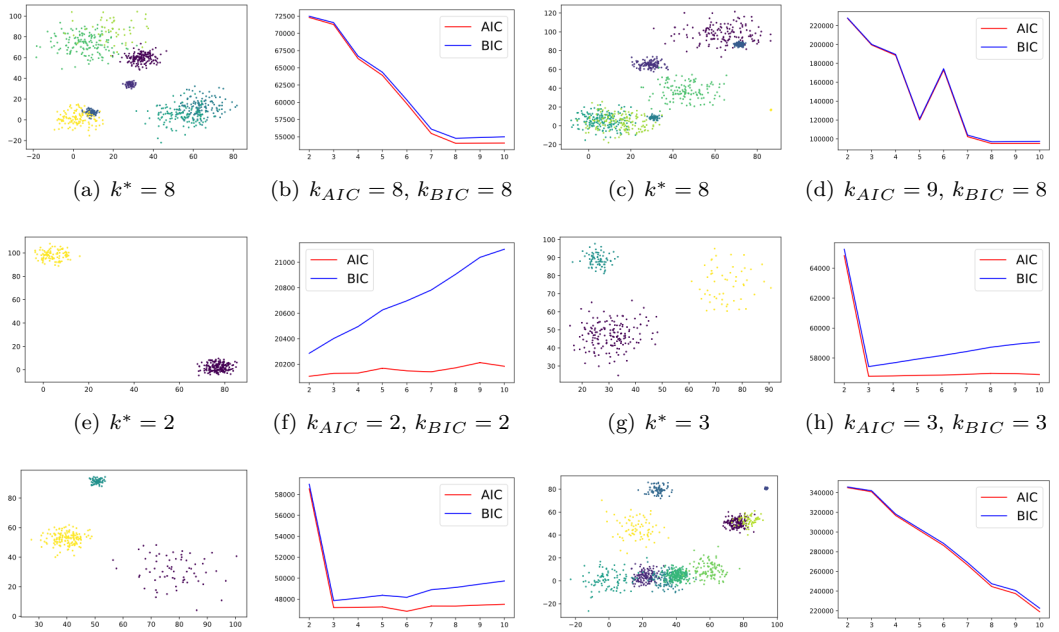
I randomly generate 1000 test datasets, and the accuracy result is shown below

Criterion	Accuracy	running time
AIC	0.9031	1249.3735
BIC	0.9452	

Table 2: The accuracy of $k_{AIC}(k_{BIC}) = k^*$, running the program 10000 times. Because AIC and BIC runs in parallel, the time may be slightly higher than single one running time

Visualization

For better describing the experimental results, I visualized the dataset and plot the AIC, BIC curves. Note, most of the data points contain more than 2 dimension, but for visualization, I only choose two of all dimensions to show. **Therefore you may find result unreasonable if you consider data has only two dimension.**



(a) $k^* = 8$ (b) $k_{AIC} = 8, k_{BIC} = 8$ (c) $k^* = 8$ (d) $k_{AIC} = 9, k_{BIC} = 8$

(e) $k^* = 2$ (f) $k_{AIC} = 2, k_{BIC} = 2$ (g) $k^* = 3$ (h) $k_{AIC} = 3, k_{BIC} = 3$

(i) $k^* = 3$ (j) $k_{AIC} = 6, k_{BIC} = 3$ (k) $k^* = 10$ (l) $k_{AIC} = 10, k_{BIC} = 10$

Figure 2: Some AIC and BIC example visualization

Comparison between AIC and BIC

- **Time complexity:** almost the same.
- **Accuracy of getting k^* :** BIC is much better.

3.3 VBEM

Variational inference is an extension of expectation-maximization that maximizes a lower bound on model evidence (including priors) instead of data likelihood. The principle behind variational methods is the same as expectation-maximization (that is both are iterative algorithms that alternate between finding the probabilities for each point to be generated by each mixture and fitting the mixture to these assigned points), but variational methods add regularization by integrating

information from prior distributions. This avoids the singularities often found in expectation-maximization solutions but introduces some subtle biases to the model. Inference is often notably slower, but not usually as much so as to render usage unpractical.

Due to its Bayesian nature, the variational algorithm needs more hyper- parameters than expectation-maximization, the most important of these being the concentration parameter weight concentration prior. Specifying a low value for the concentration prior will make the model put most of the weight on few components set the remaining components weights very close to zero. High values of the concentration prior will allow a larger number of components to be active in the mixture.

when weight concentration prior is small enough and the number of components is larger than what is found necessary by the model, the Variational Bayesian mixture model has a natural tendency to set some mixture weights values close to zero. This makes it possible to let the model choose a suitable number of effective components automatically. Only an upper bound of this number needs to be provided.

3.4 Result

I run the program 10000 times and get an averaged time cost and the accuracy. Integrating the VBEM result with the results of the previous table, I get the following table, which shows that VBEM performs much better in accuracy for k.

Criterion	Accuracy	running time
AIC	0.9031	1249.3735
BIC	0.9452	
VBEM	0.9789	254.3914

Table 3: The accuracy of $k_{AIC}(k_{BIC}) = k^*$ and running time, running the program 10000 times. **Note that the running time of AIC and BIC is linearly dependent on the range of candidate** k .

Whether running time or accuracy of $\hat{k} = k^*$, VBEM performs better. Note, AIC and BIC must fit several times to determine the best k, which makes time-consuming. Surely general EM with inital k will absolutely run more fast than VBEM. (This result will be also shown by experimental data later)

By comparing with the result of AIC and BIC. We can see that AIC or BIC is more time-consuming in finding the best k. And meanwhile VBEM has the higher accuracy for finding the best k, with a better fitting result. However, as we all know, VBEM is slower than general EM, because EM has an initial k while VBEM needs to evaluating the lower bound or something like that to automatically select k^*.

3.5 The control variable method, testing the specific data sets

Here I list some datesets which is randomly generated by changing the factors in Table 1. And I get the k in different algorithm.

No.	n_samples	n_dimension	n_component	K_{AIC}	K_{BIC}	K_{VBEM}
1	200	2	2	3	2	2
2	1000	2	2	2	2	2
3	1000	2	10	10	9	6
4	2000	2	6	7	7	6
5	2000	2	10	10	10	9
6	2000	15	6	6	6	6
7	2000	40	6	6	6	6
8	1000	15	2	2	2	2
9	1000	15	6	6	6	6
10	1000	15	10	9	9	10
11	1000	40	10	10	10	10
12	2000	40	10	9	9	10

Table 4: Performance comparison in specific datasets

Analysis

Table 4 contains three factors. Note that a generated dataset is not only decided by these three factors, but also many other factors, such as the specific number of samples of each component, the distance between each component(distribution). However, these three factors that has been listed can be called the most important factors.

- From dataset $No.1, 2$, we can see that when number of samples is small, AIC performs not very well compared with the other two methods.
- From dataset $No.4, 6, 7$, when number of dimension is small, VBEM gets a wrong k while AIC and BIC perform well.
- From dataset $No.7, 12$ or dataset $No.8, 9, 10$, VBEM is not sensitive to the number of component. That is, when number of component becomes large, VBEM performs still very well, while AIC or BIC gets a bias.

3.6 Conclusion

1. VBEM performs better than AIC and BIC In a GMM that we don't know information about at all from Table 1. That is, Given a random GMM, we prefer VBEM because VBEM has a higher accuracy and less running time.
2. When the number of dimension is small, we perfer AIC or BIC. Besides, BIC performs better in almost all situations. Therefore, we tend to choose BIC.
3. When the number of samples is small, we tend to abandon AIC because AIC seems to perform worse than the other methods in this situation.
4. VBEM is more robust over the change of number of component. when the number of component is large, we prefer VBEM to select the best k.
5. In most situations, three methods leads the totally same k.

4 Final

Finally, I'd like to express my gratitude to Prof. Tu and TAs, who help me a lot in learning machine learning. This homework report also stimulated my great interest in the research of machine learning.