Machine Learning Homework 1*

Hongzhou Liu
517030910214
deanlhz@sjtu.edu.cn

1 k-mean algorithm

For step 1, consider an arbitrary point $\boldsymbol{x_t}$ and the cluster k it belongs to before step 1, i.e. $r_{tk} = 1$. It will be assigned to the nearest $\boldsymbol{\mu_i}$ in step 1.
Assume $i = k'$ for $\boldsymbol{x_t}$, then

$$\|\boldsymbol{x_t} - \boldsymbol{\mu_{k'}}\|^2 \leq \|\boldsymbol{x_t} - \boldsymbol{\mu_k}\|^2$$

Also, $r_{tk'} = 1$ while $r_{tj} = 0, \forall j \in \{1, \cdots, K\} - \{k'\}$.

Thus,

$$J'(\boldsymbol{\mu_1}, \cdots, \boldsymbol{\mu_k}) = \sum_{t=1}^{N}\sum_{k'=1}^{K} r_{tk'} \|\boldsymbol{x_t} - \boldsymbol{\mu_{k'}}\|^2 \leq J(\boldsymbol{\mu_1}, \cdots, \boldsymbol{\mu_k}) = \sum_{t=1}^{N}\sum_{k=1}^{K} r_{tk} \|\boldsymbol{x_t} - \boldsymbol{\mu_k}\|^2$$

then step 1 will never increase the objective function.

For step 2, in this step, r_{nk} will not be changed. The algorithm will compute new average values for each cluster. It is equivalent to prove that if $\boldsymbol{\mu_i}$ is the average point of cluster i then $\sum_{t=1}^{N} r_{ti} \|\boldsymbol{x_t} - \boldsymbol{\mu_i}\|^2$ is minimized. Denote cluster i as C_i, we want $\min_{\boldsymbol{\mu_i}} f(\boldsymbol{\mu_i}) = \sum_{\boldsymbol{x}\in C_i} \|\boldsymbol{x} - \boldsymbol{\mu_i}\|^2$. Let $\dfrac{\partial f}{\partial \boldsymbol{\mu_i}} = 0$, then

$$\frac{\partial f}{\partial \boldsymbol{\mu_i}} = \frac{\partial}{\partial \boldsymbol{\mu_i}} \sum_{\boldsymbol{x}\in C_i} (\boldsymbol{\mu_i}^T\boldsymbol{\mu_i} - 2\boldsymbol{\mu_i}^T\boldsymbol{x} + \boldsymbol{x}^T\boldsymbol{x}) = \sum_{\boldsymbol{x}\in C_i} 2(\boldsymbol{\mu_i} - \boldsymbol{x}) = 0$$

Thus,

$$\boldsymbol{\mu_i} = \frac{1}{|C_i|}\sum_{\boldsymbol{x}\in C_i} \boldsymbol{x}$$

which is the average point of cluster i. In this case, the object function will also not be increased.

2 k-mean vs GMM

We've learned the differences between k-mean and GMM in the class and we know how to degenerate GMM to k-mean. According to the notes I took in the class, there're some ways to design such variant algorithm.

- Fix covariance matrix $\boldsymbol{\Sigma_k}$ in GMM as $\boldsymbol{\Sigma_k} = \epsilon_k \boldsymbol{I}$ or a diagonal matrix
- Fix π_k for every cluster k in GMM
- Use soft-cut in k-mean
- Use Mahalanobis distance in k-mean

*GitHub repo: https://github.com/DeanAlkene/CS420-MachineLearning/tree/master/A1

Here, we design a variant algorithm by using soft-cut in k-mean. To achieve "soft", we must re-define r_{tk} as the probability of $\boldsymbol{x_t}$ being assigned to cluster k where $\sum_{k=1}^{K} r_{tk} = 1$. Noticing that softmax function is able to take an input and normalize it into a probability distribution. Thus, we can define r_{tk} as

$$r_{tk} = \frac{e^{\|\boldsymbol{x_t} - \boldsymbol{\mu_k}\|^2}}{\sum_{i=1}^{K} e^{\|\boldsymbol{x_t} - \boldsymbol{\mu_i}\|^2}}$$

.

Then, in the M-step, we want to

$$\min J(\boldsymbol{\mu_1}, \cdots, \boldsymbol{\mu_k}) = \min \sum_{t=1}^{N} \sum_{k=1}^{K} r_{tk} \|\boldsymbol{x_t} - \boldsymbol{\mu_k}\|^2$$

For each $i \in \{1, \cdots, K\}$, let

$$\frac{\partial}{\partial \boldsymbol{\mu_i}} J(\boldsymbol{\mu_1}, \cdots, \boldsymbol{\mu_k}) = \frac{\partial}{\partial \boldsymbol{\mu_i}} \sum_{t=1}^{N} r_{ti} \|\boldsymbol{x_t} - \boldsymbol{\mu_i}\|^2 = 0$$

Thus,

$$2 \sum_{t=1}^{N} r_{ti} (\boldsymbol{\mu_i} - \boldsymbol{x_t}) = 0$$

And,

$$\boldsymbol{\mu_i} = \frac{\sum_{t=1}^{N} r_{ti} \boldsymbol{x_t}}{\sum_{t=1}^{N} r_{ti}}$$

Algorithm 1: Soft Clustering K-mean

Input : The number of clusters K
Output : $r_{tk}, \boldsymbol{\mu_k}(k = 1, \cdots, K, t = 1, \cdots, N)$

Initialize $\boldsymbol{\mu_1}, \cdots, \boldsymbol{\mu_K}$ and evaluate $J(\boldsymbol{\mu_1}, \cdots, \boldsymbol{\mu_K})$
while *the convergence criterion is* ***not*** *satisfied* **do**
    E-step: Evaluate the responsibilities using the current parameter values
    for $t \leftarrow 1$ *to* N **do**
        for $k \leftarrow 1$ *to* K **do**
            $r_{tk} \leftarrow \frac{e^{\|\boldsymbol{x_t} - \boldsymbol{\mu_k}\|^2}}{\sum_{i=1}^{K} e^{\|\boldsymbol{x_t} - \boldsymbol{\mu_i}\|^2}}$
    M-step: Re-estimate the parameters using the current responsibilities
    for $i \leftarrow 1$ *to* K **do**
        $\boldsymbol{\mu_i} \leftarrow \frac{\sum_{t=1}^{N} r_{ti} \boldsymbol{x_t}}{\sum_{t=1}^{N} r_{ti}}$
return $r_{tk}, \boldsymbol{\mu_k}(k = 1, \cdots, K, t = 1, \cdots, N)$

Advantage:

- The variant uses soft clustering, which is much more robust than original k-mean algorithm using hard clustering.

Limitation: As a specialized EM algorithm, our algorithm inevitably has some limitations

- Local optima problem: we may not find the global optima
- Sensitivity of initial values
- Hardness of determining k

3 k-mean vs CL

Similarities:

- Hard clustering
- The performance highly relies on initialization
- Using distance metrics instead of probability
- Cannot estimate the number of clusters

Differences:

- CL is a kind of on-line learning, while k-means is a kind of batch learning
- CL is more computational efficient when dataset is large
- CL has an additional hyperparameter η
- K-mean is easy to understand and implement, however, CL is more complicated

If we want to automatically determine the number of clusters in k-mean using RPCL, we may preprocess dataset using RPCL, find the k and then apply k-mean. Thus, we can regard centers as rivals and apply RPCL on the dataset. Then, we delete those losers and the winners are the initial centers of K-means. The detailed algorithm are shown as:

Algorithm 2: RPCL for Selecting Best k

Input : The dataset $\boldsymbol{x_1}, \cdots, \boldsymbol{x_N}$, Max number of clusters K_{max}, Update rate η, Threshold θ
Output : The best k k^*, Centers $\boldsymbol{\mu_k}, (k = 1, \cdots, k^*)$

Initialize $\boldsymbol{\mu_1}, \cdots, \boldsymbol{\mu_K}$ and evaluate
for $epoch \leftarrow 1$ *to* max_iter **do**
  for *data* $\boldsymbol{x_n}$ *in dataset* **do**
    Calculate Euclidean distance to $\boldsymbol{\mu_i}, \cdots, \boldsymbol{\mu_K}$
    Calculate $p_{n,k}$

$$p_{n,k} = \begin{cases} 1 & \text{if } k = c = argmin_k \epsilon_n(\theta_k) \\ -\gamma & \text{if } k = r = argmin_{k \neq c} \epsilon_n(\theta_k) \\ 0 & \text{otherwise} \end{cases}$$

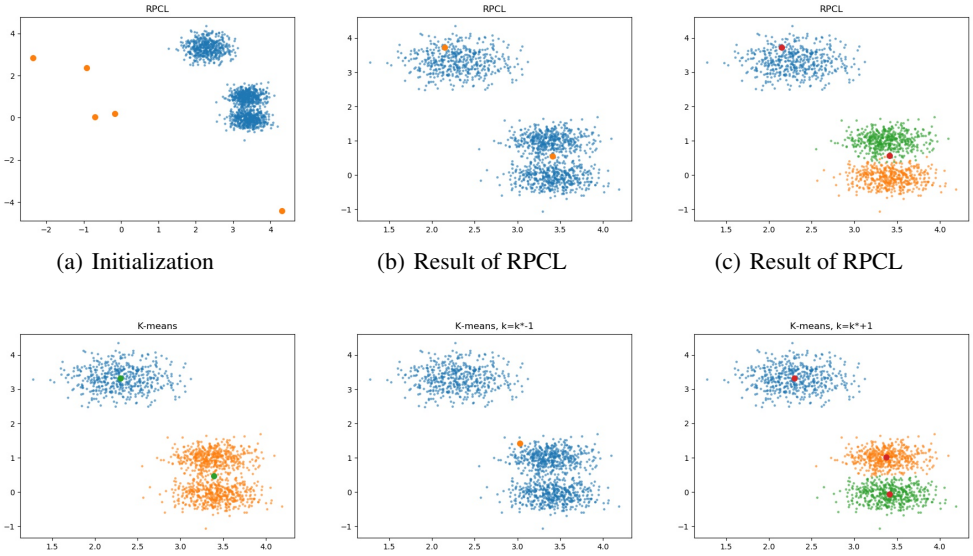
    Update $\boldsymbol{\mu_k}$

$$\boldsymbol{\mu_k} \leftarrow \boldsymbol{\mu_k} + p_{n,k}\eta(\boldsymbol{x_n} - \boldsymbol{\mu_k})$$

for $k \leftarrow 0$ *to* K_{max} **do**
  $\boldsymbol{x'_k} \leftarrow$ the closet point to $\boldsymbol{\mu_k}$
  if $\left\|\boldsymbol{x'_k} - \boldsymbol{\mu_k}\right\| \geq \theta$ **then**
    Remove $\boldsymbol{\mu_k}$
$k^* \leftarrow$ The number of remaining centers
return $k^*, \boldsymbol{\mu_k}, (k = 1, \cdots, k^*)$

I implemented the algorithm in Python, and compared the performance on different datasets. As we can see, RPCL is a quite good auxiliary method for clustering. Here're some observations:

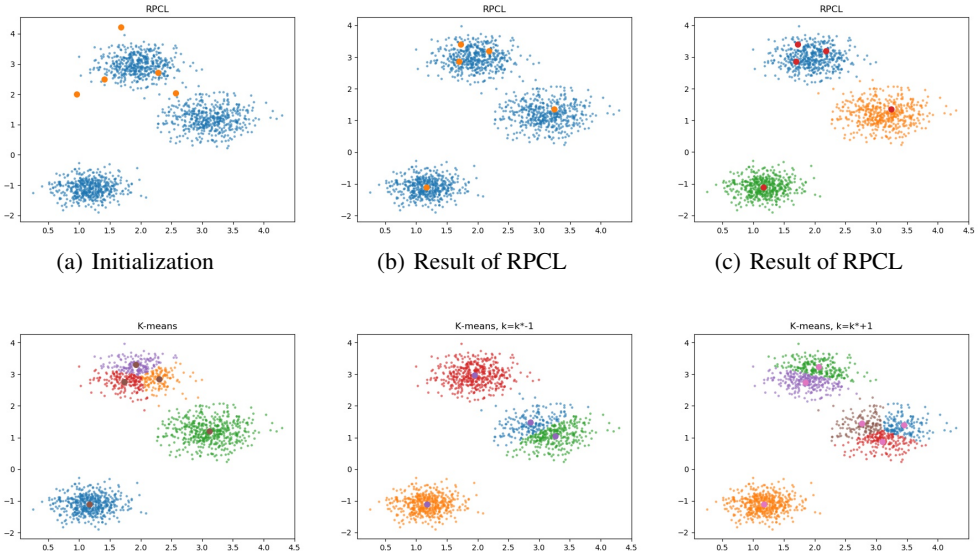
- In Experiment 1, the right two clusters are close to each other and the initial position of centers are away from the dataset. And the result shows that our k^* is less than real k^*.
- In Experiment 2, the initial centers are close to one specific cluster. At last, the upper left cluster is divided into 3 clusters.
- In Experiment 3, the result is quite perfect.



(a) Initialization (b) Result of RPCL (c) Result of RPCL

(d) K-means on k^* (e) K-means on $k^* - 1$ (f) K-means on $k^* + 1$

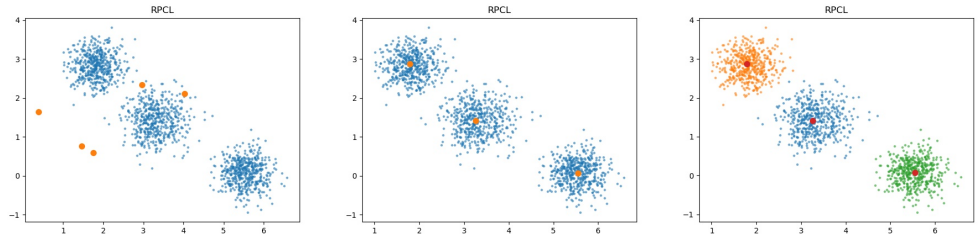
Figure 1: Experiment 1



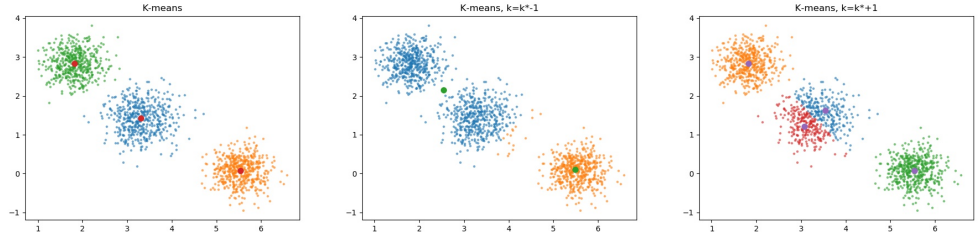
(a) Initialization (b) Result of RPCL (c) Result of RPCL

(d) K-means on k^* (e) K-means on $k^* - 1$ (f) K-means on $k^* + 1$

Figure 2: Experiment 2



(a) Initialization (b) Result of RPCL (c) Result of RPCL



(d) K-means on k^* (e) K-means on $k^* - 1$ (f) K-means on $k^* + 1$

Figure 3: Experiment 3

It shows that RPCL is sensitive to initialization. And the result also depends on the distribution of data. I thought there're still some ways to improve it:

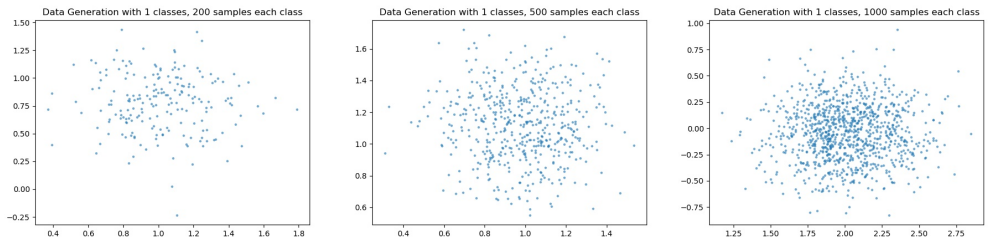
- Carefully initialize centers by observing distribution of dataset first
- Avoid "crowded" initial centers
- Use improved RPCL algorithm[1]

4 model selection of GMM

I implemented AIC, BIC on GMM and VBEM, then tested them on different datasets. Their performance differed slightly when the number of clusters and the number of samples changed.

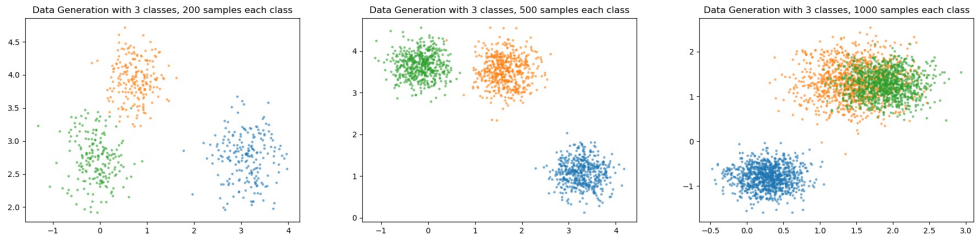
4.1 Baseline

I generated datasets with cluster numbers in range (1, 3, 8) and sample sizes in range (200, 500, 1000) for each cluster and assigned labels for each cluster. Here are the visualization on those datasets.



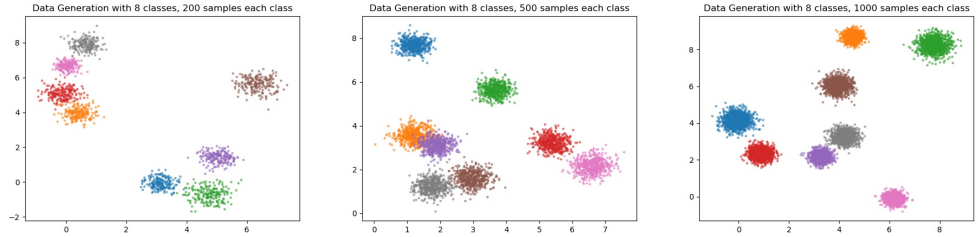
(a) sample_size=200 (b) sample_size=500 (c) sample_size=1000

Figure 4: cluster_number=1



(a) sample_size=200 (b) sample_size=500 (c) sample_size=1000

Figure 5: cluster_number=3

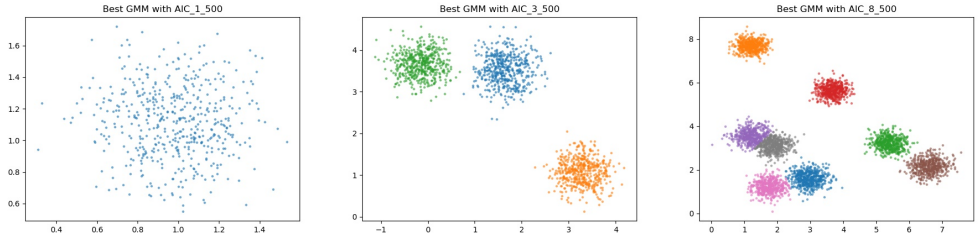


(a) sample_size=200 (b) sample_size=500 (c) sample_size=1000

Figure 6: cluster_number=8

4.2 Experiment on Different Cluster Numbers

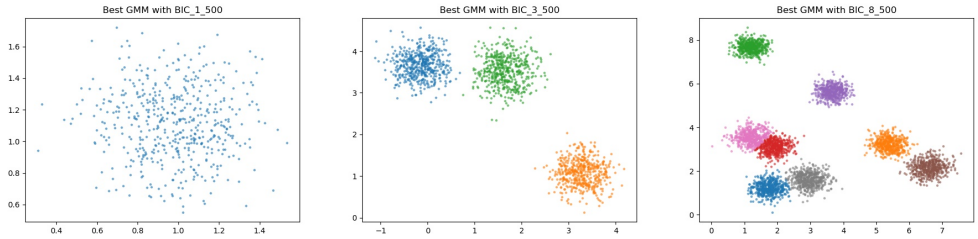
In this part, we fix sample size and observe results of AIC, BIC and VBEM on different cluster numbers.



(a) cluster_number=1 (b) cluster_number=3 (c) cluster_number=8

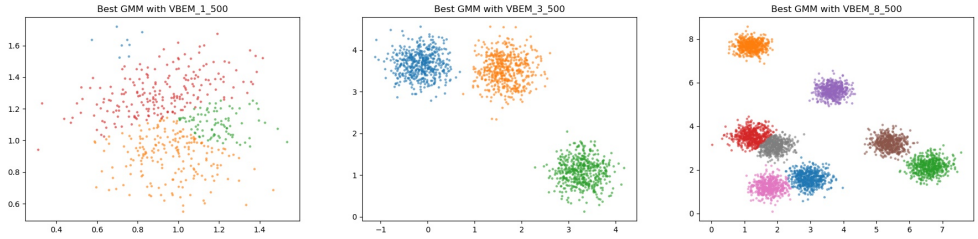
Figure 7: AIC, sample_size=500

The results shows that, when the cluster numbers get large, the performance of AIC, BIC and VBEM are the same, and they all performed well. When there are only one cluster, VBEM may divide the dataset in to more than one clusters. It is due to the design of VBEM, which initialize with a lot of Gaussian distributions. Also, the sample sizes of each clusters affect the result. If we decrease the sample_size, AIC and BIC may sometimes also encounter the problem during the experiment. If we increses the sample_size, the problem resolves.



(a) cluster_number=1 (b) cluster_number=3 (c) cluster_number=8

Figure 8: BIC, sample_size=500

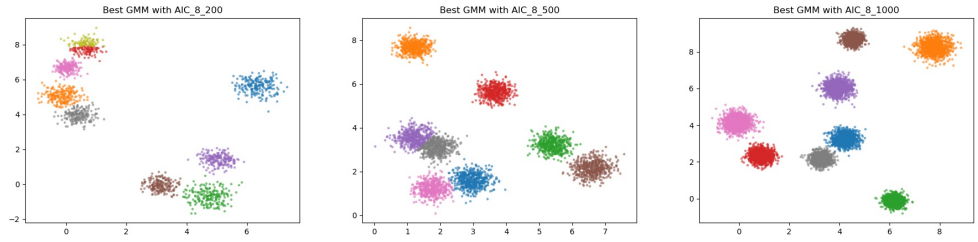


(a) cluster_number=1 (b) cluster_number=3 (c) cluster_number=8

Figure 9: VBEM, sample_size=500

4.3 Experiment on Different Sample Sizes

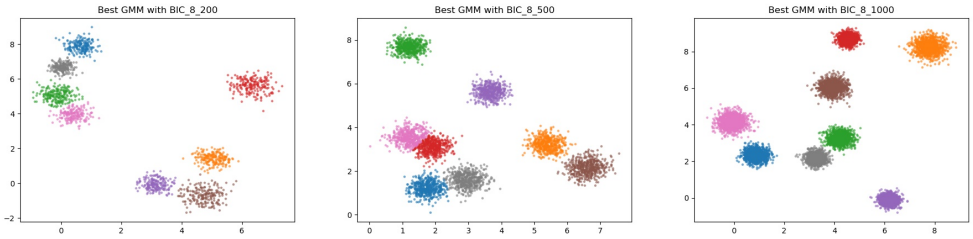
In this section, we fix cluster number and investigate the effect of AIC, BIC, VBEM on different sample sizes.



(a) sample_size=200 (b) sample_size=500 (c) sample_size=1000

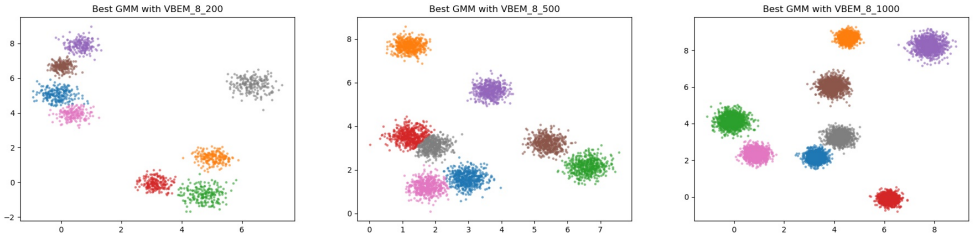
Figure 10: AIC, cluster_number=8

AIC, BIC and VBEM all performed quite well if the number of samples is sufficient. However, if we decrease the sample_size, AIC has some problems. BIC performs better in this situation because it not only considers the number of free parameters but also the sample size. Additionally, if the number of clusters decreases, we may not observe the difference among those methods.



(a) sample_size=200 (b) sample_size=500 (c) sample_size=1000

Figure 11: BIC, cluster_number=8



(a) sample_size=200 (b) sample_size=500 (c) sample_size=1000

Figure 12: VBEM, cluster_number=8

References

[1] J. Yang and L. Jin, "An Improved RPCL Algorithm for Determining Clustering Number Automatically," TENCON 2006 - 2006 IEEE Region 10 Conference, Hong Kong, 2006, pp. 1-3.