Week 20

Self-Adaptive Multiprototype-Based Competitive Learning Approach: A k-Means-Type Algorithm for Imbalanced Data Clustering

Clustering on imbalanced data is a totally different and more challenging problem. Its difficulty is basically on two aspects:
1) the imbalance status, that is, the number of minority clusters and the imbalance ratio, is unknown
2) it is more difficult to determine the number of clusters for imbalanced data. A common clustering result on imbalanced clusters is that the minority cluster is either merged into the majority cluster as one cluster or treated as noises and outliers.

k-means tends to produce balanced clustering result.

A. Selection of Number of Prototypes
Each cluster can be represented by more than one seed point (also called prototype interchangeably) and made up by multiple subclusters. For imbalanced data, the majority cluster tends to have more seed points and the minority cluster has less.
However, a key problem related to multi-prototype is that how many prototypes are needed to well represent the entire dataset.

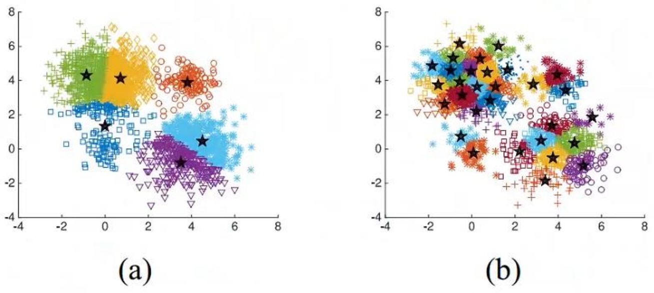


Fig. 3. Example of four imbalanced clusters with size [61, 1212, 606, 121]. The clustering result is generated by adaptive *k*-means with (a) 6 seed points and (b) 30 seed points.

Therefore, we propose PNS to determine the number of seed points in a self-adaptive manner. The seed points are gradually added by the algorithm until one seed point is driven away by the rival penalization mechanism.

$$\mathbf{m}_j(t+1) = \begin{cases} \mathbf{m}_j(t) + \alpha_c(\mathbf{x}_t - \mathbf{m}_j(t)) & \text{if } I_{j,\mathbf{x}_t} = 1 \\ \mathbf{m}_j(t) - \eta\beta_j\alpha_c(\mathbf{x}_t - \mathbf{m}_j(t)) & \text{otherwise} \end{cases}$$

$$\beta_j = \exp\left(-\frac{\|\mathbf{m}_j - \mathbf{x}_t\|^2 - \|\mathbf{m}_c - \mathbf{x}_t\|^2}{\|\mathbf{m}_c - \mathbf{m}_j\|^2}\right)$$

and **m**c is the winner seed point. β_j makes the rival penalization large if **x**t is close to the middle line between **m**j and **m**c because the points are more dense if they are between two seed points in the same cluster.

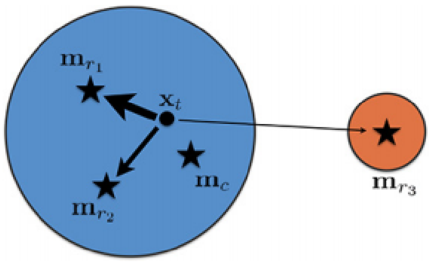


Fig. 4. Example of two imbalanced classes with four seed points. $\mathbf{m}_c$ is the winner of $\mathbf{x}_t$. $\mathbf{m}_{r_1}$, $\mathbf{m}_{r_2}$, and $\mathbf{m}_{r_3}$ are three rivals. The majority cluster (in blue) is shared by three seed points and the minority cluster (in red) is represented by one seed point. The thickness of the arrow indicates the value of β_j.

When there are enough number of seed points added, the first priority is to drive one of the seed points in the majority cluster away.

If the modified RPCCL converges and there is no seed point driven away, one new seed point is added.
Local density for each cluster member **x**i in the jth subcluster *Cj*

$$\rho_i^j = \sum_{i' \in \mathcal{C}_j} [\![d(\mathbf{x}_i, \mathbf{x}_{i'}) < \epsilon]\!]$$

The density gap of *Cj*

$$\delta_j = \max_{i \in \mathcal{C}_j} \min_{i' \in \mathcal{C}_j : \rho_{i'} > \rho_i} \frac{d(\mathbf{x}_i, \mathbf{x}_{i'})}{\bar{d}_j}$$

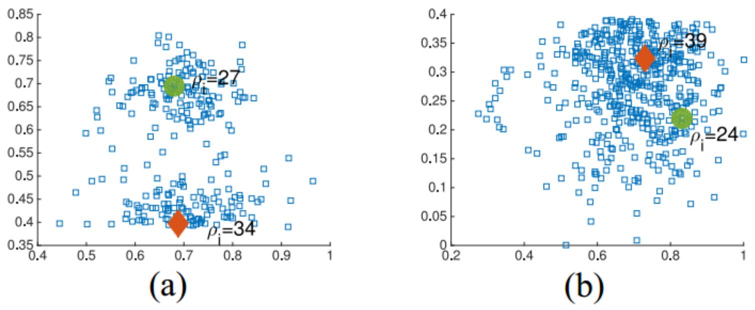


Fig. 5. Two subclusters with (a) density gap and (b) no obvious density gap. The red diamond point is the density peak with the highest local density and the green circle point is the density peak that it has the largest distance to the point with higher local density. ρ_i is the local density.

The case in Fig. 5(a) has higher priority to be selected as the subcluster to add new seed point.
The new subcluster to be split by the new seed point is selected by

$$j^* = \arg\max_{j=1,\dots,K} n_j \delta_j.$$

Where n_j is the number of winning times.
As the number of seed points K is increasing, the averaged number of winning times N/K for each seed point is decreasing. Thus, the moving magnitude of the seed points is decreasing as more seed points are added. To avoid the algorithm terminating due to this reason

$$\mathbf{m}_j(t+1) = \begin{cases} \mathbf{m}_j(t) + K\alpha_c\big(\mathbf{x}_t - \mathbf{m}_j(t)\big) & \text{if } I_{j,\mathbf{x}_t} = 1 \\ \mathbf{m}_j(t) - K\eta\beta_j\alpha_c\big(\mathbf{x}_t - \mathbf{m}_j(t)\big) & \text{otherwise.} \end{cases}$$

B. Subcluster Grouping With Model Selection

Once PNS terminates when at least one seed points are driven away, the number of remaining seed points $\widehat{K}$ is used to represent the data, namely, $\widehat{K}$ subclusters.
The objective of SGMS is to merge the subclusters in the same cluster first. The subclusters with low separation measure should be merged first because they are probably in the same cluster.
Project into the line between μi and μj

$$x' = \frac{(\mathbf{x} - \boldsymbol{\mu}_0)^T(\boldsymbol{\mu}_i - \boldsymbol{\mu}_j)}{\|\boldsymbol{\mu}_i - \boldsymbol{\mu}_j\|^2}$$

where $\boldsymbol{\mu}_i$ and $\boldsymbol{\mu}_j$ are the centers of $\mathcal{C}_i$ and $\mathcal{C}_j$, and $\boldsymbol{\mu}_0 = (\boldsymbol{\mu}_i + \boldsymbol{\mu}_j)/2$ is the middle point between two centers. The projected members are 1-D points. The centers $\boldsymbol{\mu}_i$ and $\boldsymbol{\mu}_i$ are projected to the position -0.5 and 0.5, respectively. Thus, we can obtain two sets of 1-D points by projecting the cluster members from $\mathcal{C}_i$ and $\mathcal{C}_j$. The mean and variance of the projected points, μ_i, μ_j, σ_i^2, and σ_j^2 can then be calculated. Therefore, the binary Gaussian mixture probability density function is written as

$$f(u) = \frac{|\mathcal{C}_i|}{|\mathcal{C}_i| + |\mathcal{C}_j|} p\Big(u_i | 0.5, \sigma_i^2\Big) + \frac{|\mathcal{C}_i|}{|\mathcal{C}_i| + |\mathcal{C}_j|} p\Big(u_j | -0.5, \sigma_j^2\Big)$$

Then, we use an interval with step 0.01 from −0.5 to 0.5 A = {−0.5, −0.49,..., 0.49, 0.5} to calculate the discrete probability densities f(A).

$$s_{ij} = \frac{1}{\min f(\mathcal{A})}.$$

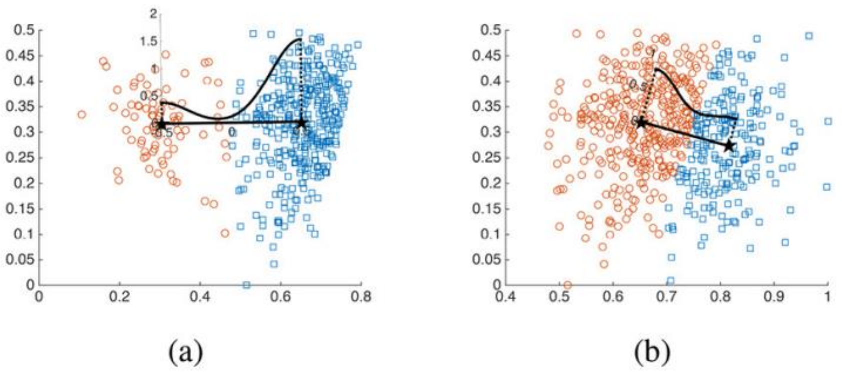
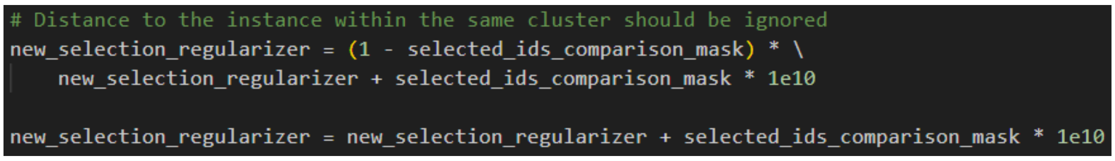


Fig. 6. Example of calculating separation measure by using 1-d Gaussian mixture probability density function. (a) Two subclusters belong to different clusters. $s_{ij} = 13.37$. (b) Two subclusters belong to the same cluster. $s_{ij} = 2.86$.

Regularization



```
# Distance to the instance within the same cluster should be ignored
new_selection_regularizer = (1 - selected_ids_comparison_mask) * \
    new_selection_regularizer + selected_ids_comparison_mask * 1e10

new_selection_regularizer = new_selection_regularizer + selected_ids_comparison_mask * 1e10
```

USL: the instance within the same cluster should be ignored

Ours: consider all selected instances