A Survey of Kernel Clustering Methods

Maurizio Filippone, Francesco Camastra, Francesco Masulli and Stefano Rovetta

Presented by: Kedar Grama
Outline

• Unsupervised Learning and Clustering
• Types of clustering algorithms
• Clustering Algorithms
  - Partitioning Methods (in Euclidean Space)
    • K-Means
    • Self Organizing Maps (SOM)
    • Neural Gas
    • Fuzzy C-Means
    • Probabilistic C-Means
  - Kernel Clustering Methods
    • Kernel K-Means
    • Kernel SOM
    • Kernel Neural Gas
    • Kernel Fuzzy C-Means
    • Kernel Probabilistic C-Means
    • One class SVMs and Support Vector Clustering
Unsupervised Learning And Clustering

• Supervised learning - human effort involved
  • Example: Learning conditional distribution $P(Y|X)$, $X$: features, $Y$: classes

• Unsupervised learning - no human effort involved
  • Example: learning distribution $P(X)$, $X$: features

• Definition: **Clustering** is the task of grouping a set of objects in such that objects in the same group are more similar to each other than to those in other groups
Types of Clustering Algorithms

Clustering Algorithms

Flat Algorithms
- Hard Partitioning
  - Examples:
    • K-Means
    • Self Organizing maps
    • DBSCAN
- Soft Partitioning
  - Examples:
    • Expectation Maximization
    • Fuzzy Clustering Methods

Hierarchical Algorithms
- Single Linkage
- Complete Linkage
- Other Linkages
K-means

• Objective: Minimize the empirical quantization error $E(X)$

$$E(X) = \frac{1}{2n} \sum_{i=1}^{k} \sum_{x \in \pi_i} \|x - v_i\|^2$$

• Algorithm:
  1. choose the number $k$ of clusters;
  2. initialize the codebook $V$ with vectors randomly picked from $X$;
  3. compute the Voronoi set $i$ associated to the code vector $v_i$;
  4. move each code vector to the mean of its Voronoi set

$$v_i = \frac{1}{|\pi_i|} \sum_{x \in \pi_i} x$$

  5. return to step 3 if any code vector has changed otherwise
  6. return the codebook.
K-means Visualization

Kernel Clustering Basics

• Mercer Kernels:
  • Polynomial:
    \[ K^{(p)}(x_i, x_j) = (1 + x_i \cdot x_j)^p, \quad p \in \mathbb{N} \]
  • Gaussian:
    \[ K^{(g)}(x_i, x_j) = \exp \left( -\frac{||x_i - x_j||^2}{2\sigma^2} \right), \quad \sigma \in \mathbb{R}. \]

• Distances in kernel space can be computed by using the distance kernel trick
  \[
  \|\Phi(x_i) - \Phi(x_j)\|^2 \\
  = (\Phi(x_i) - \Phi(x_j)) \cdot (\Phi(x_i) - \Phi(x_j)) \\
  = \Phi(x_i) \cdot \Phi(x_i) + \Phi(x_j) \cdot \Phi(x_j) - 2\Phi(x_i) \cdot \Phi(x_j) \\
  = K(x_i, x_i) + K(x_j, x_j) - 2K(x_i, x_j)
  \]

• First map the data set \(X\) into kernel space by computing the Gram Matrix, \(K\), where each element \(k_{ij}\) is the dot product in kernel space.
  \[ \Phi(x_i) \cdot \Phi(x_i) \text{ using } \|\Phi(x_i) - \Phi(x_j)\|^2 = k_{ii} + k_{jj} - 2k_{ij} \]
Kernel K-means

- The Voronoi region and Voronoi Set in the feature space are redefined as: \( R_i^\Phi = \{ x^\Phi \in \mathcal{F} | i = \arg \min_j \| x^\Phi - v_j^\Phi \| \} \) and \( \pi_i^\Phi = \{ x \in X | i = \arg \min_j \| \Phi(x) - v_j^\Phi \| \} \)

- Algorithm:
  1. Project the data set \( X \) into a feature space \( \mathcal{F} \), by means of a nonlinear mapping \( \Phi \)
  2. Initialize the codebook \( V^\Phi = (v_1^\Phi, \ldots, v_k^\Phi) \) with \( v_i^\Phi \in \mathcal{F} \)
  3. Compute for each center the set \( v_i^\Phi \) the set \( \pi_i^\Phi \)
  4. Update the code vectors \( v_i^\Phi \) in \( \mathcal{F} \):
     \[ v_i^\Phi = \frac{1}{|\pi_i^\Phi|} \sum_{x \in \pi_i^\Phi} \Phi(x) \]
  5. Go to step 3 until any \( v_i^\Phi \) changes
  6. Return the feature space codebook.
Kernel K-means Continued

• Since $\Phi$ is not explicitly known updating the code vectors is not straightforward

• Writing each centroid in Kernel space $v_j^\Phi = \sum_{h=1}^{n} \gamma_{jh} \phi(x_h)$ where $\gamma_{jh}$ is 1 if $x_h$ belongs to the set $j$, zero otherwise.

• Now, $\|\phi(x_i) - v_j^\Phi\|^2 = \|\phi(x_i) - \sum_{h=1}^{n} \gamma_{jh} \phi(x_h)\|^2$ can be expanded to:

$$\left\| \phi(x_i) - \sum_{h=1}^{n} \gamma_{jh} \phi(x_h) \right\|^2 = k_{ii} - 2 \sum_{h} \gamma_{jh} k_{ih} + \sum_{r} \sum_{s} \gamma_{jr} \gamma_{js} k_{rs}$$

• Gram Matrix, ideally has a block diagonal structure if the clusters are uniformly dense and hence provide a good way to estimate the number of clusters too.
Kernel K-means Examples

Iris Data

Performance

Eigenvalues of Gram Mat with RBF = 0.5 showing three major clusters

Self Organizing Map (SOM)

- Code vectors organized on a grid and their adaptation is propagated along the grid

- Some popular metrics for the map include the Manhattan distance where the distance between two elements \( r = (r_1, r_2) \) and \( s = (s_1, s_2) \) is:

\[
d_{rs} = |r_1 - s_1| + |r_2 - s_2|
\]

- Algorithm
  1. Initialize the codebook \( V \) randomly picking from \( X \)
  2. Initialize the set \( C \) of connections to form the rectangular grid of dimension \( n_1 \times n_2 \)
  3. Initialize \( t = 0 \)
  4. Randomly pick an input \( x \) from \( X \).
  5. Determine the winner: \( s(x) = \arg \min_{v_j \in V} \| x - v_j \| \)
  6. Adapt each code vector: \( \Delta v_j = \varepsilon(t) h(d_{rs}) (x - v_j) \)

\[
h(d_{rs}) = \exp\left( -\frac{d_{rs}^2}{2\sigma^2(t)} \right) \quad \sigma(t) = \sigma_i \left( \frac{\sigma_f}{\sigma_i} \right)^{t/t_{max}}
\]

\[
\varepsilon(t) = \varepsilon_i \left( \frac{\varepsilon_f}{\varepsilon_i} \right)^{t/t_{max}}
\]
  7. Increment \( t \)
  8. If \( t < t_{max} \) go to step 4
A SOM showing U.S. Congress voting patterns. The data were initially distributed randomly on a 2D grid and then clustered. The grey dots show the neurons. The first box shows clustering and, the second distances. The third is a panel shows the party affiliation, red-republican and blue-democrat and the rest are the features, which, in this instance are yes(blue) or no(red) votes.

Kernel SOM

• Again the algorithm is adapted by first mapping the points to kernel space.
• The code vectors are defined as:

\[ v_j^\Phi = \sum_{h=1}^{n} \gamma_{jh} \Phi(x_h) \quad (1) \]

• The winner is computed with:

\[
s(\Phi(x_i)) = \arg \min_{v_j^\Phi \in V} \| \Phi(x_i) - v_j^\Phi \|
\]

or

\[
s(\Phi(x_i)) = \arg \min_{v_j^\Phi \in V} \left( k_{ii} - 2 \sum_{h} \gamma_{jh} k_{ih} \sum_{r} \sum_{s} \gamma_{jr} \gamma_{js} k_{rs} \right)
\]

• The update rules are:

\[
v_j^{\Phi'} = v_j^\Phi + \varepsilon(t) h(d_{rs})(\Phi(x) - v_j^\Phi).
\]

Using (1) we get

\[
\sum_{h=1}^{n} \gamma'_{jh} \Phi(x_h) = \sum_{h=1}^{n} \gamma_{jh} \Phi(x_h) + \varepsilon(t) h(d_{rs}) \times \left( \Phi(x) - \sum_{h=1}^{n} \gamma_{jh} \Phi(x_h) \right)
\]

\[
\gamma'_{jh} = \begin{cases} 
(1 - \varepsilon(t) h(d_{rs})) \gamma_{jh} & \text{if } i \neq j, \\
(1 - \varepsilon(t) h(d_{rs})) \gamma_{jh} + \varepsilon(t) h(d_{rs}) & \text{otherwise.}
\end{cases}
\]
Kernel SOM Example

Input data clustered by Kernel SOM on the right

Data clustered by Kernel SOM, using an RBF of 0.1 and 2 clusters

Neural Gas and Kernel Neural Gas

• Similar to SOM the major difference being a soft adaptation rule in which all neurons are adapted to each individual input.

$$\Delta v_j = \varepsilon(t) h_{\lambda}(\rho_j)(x - v_j), \varepsilon(t) \in [0, 1] \quad h_{\lambda}(\rho_j) = \exp(-\rho_j / \lambda)$$

• $\rho_j$ is the rank of closeness of the current code vector $j$, to the input $x$

• $\lambda$ is the characteristic decay

• For the kernelized version the update rule is:

$$\Delta v_j^\Phi = \varepsilon h_{\lambda}(\rho_j)(\Phi(x) - v_j^\Phi)$$
Fuzzy C-Means

- Starts by defining a membership matrix, $\mathbf{A}_{cn}$ denotes vector space of $c \times n$ real matrices;

$$M_{fc} = \left\{ U \in \mathbf{A}_{cn} \left| \begin{array}{l} u_{ih} \in [0, 1] \quad \forall i, h; \sum_{i=1}^{c} u_{ih} = 1 \quad \forall h; \quad 0 < \sum_{h=1}^{n} u_{ih} < n \quad \forall i \end{array} \right. \right\}$$

- Minimizes functional:

$$J(U, V) = \sum_{h=1}^{n} \sum_{i=1}^{c} (u_{ih})^m \|x_h - v_i\|^2 \text{ with the constraint } \sum_{i=1}^{c} u_{ih} = 1 \quad \forall i = 1, \ldots, n.$$  

- $m$ controls the fuzziness of the memberships and is usually set close to 2, if $m$ tends to 1, the solution tends to the k-means solution

- Lagrangian of the objective is

$$L_h = \sum_{i=1}^{c} (u_{ih})^m \|x_h - v_i\|^2 + \alpha_h \left( \sum_{i=1}^{c} u_{ih} - 1 \right)$$

- Taking the derivative with respect to $u_{ih}$ and $v_i$ and setting them to zero yields the iteration scheme:

$$u_{ih} = \frac{\sum_{j=1}^{n} (u_{ij})^m \|x_h - v_j\|^{2/(m-1)}}{\sum_{j=1}^{n} (u_{ij})^m} \quad v_i = \frac{\sum_{h=1}^{n} (u_{ih})^m x_h}{\sum_{h=1}^{n} (u_{ih})^m}$$
Kernel Fuzzy C-Means

• The objective in the kernel space is:

\[ J^\Phi(U, V) = \sum_{h=1}^{n} \sum_{i=1}^{c} (u_{ih})^m \| \Phi(x_h) - \Phi(v_i) \|^2 \]

• In case of the Gaussian Kernel the derivative is:

\[ \frac{\partial K(x_h, v_i)}{\partial v_i} = \frac{(x_h - v_i)}{\sigma^2} K(x_h, v_i) \]

• This yields the iteration scheme:

\[ u_{ih}^{-1} = \sum_{j=1}^{c} \left( \frac{1 - K(x_h, v_i)}{1 - K(x_h, v_j)} \right)^{1/(m-1)}, \quad v_i = \frac{\sum_{h=1}^{n} (u_{ih})^m K(x_h, v_i) x_h}{\sum_{h=1}^{n} (u_{ih})^m K(x_h, v_i)} \]
Possibilistic C-Means

• Here, the class membership of a data point can be high for more than one class

• Objective that is minimized is:

\[
J(U, V) = \sum_{h=1}^{n} \sum_{i=1}^{c} (u_{ih})^m \| x_h - v_i \|^2 + \sum_{i=1}^{c} \eta_i \sum_{h=1}^{n} (1 - u_{ih})^m
\]

• The iteration scheme is:

\[
u_{ih} = \left[ 1 + \left( \frac{\| x_h - v_i \|^2}{\eta_i} \right)^{1/(m-1)} \right]^{-1}, \quad v_i = \frac{\sum_{h=1}^{n} (u_{ih})^m x_h}{\sum_{h=1}^{n} (u_{ih})^m}
\]

• For the parameter \( \eta_i \) the authors suggest using:

\[
\eta_i = \gamma \frac{\sum_{h=1}^{n} (u_{ih})^m \| x_h - v_i \|^2}{\sum_{h=1}^{n} (u_{ih})^m}
\]
Kernel Possibilistic C-Means

- Kernelization of the metric in the objective yields:
  \[ J^\Phi(U, V) = \sum_{h=1}^{n} \sum_{i=1}^{c} (u_{ih})^m \| \Phi(x_h) - \Phi(v_i) \|^2 + \sum_{i=1}^{c} \eta_i \sum_{h=1}^{n} (1 - u_{ih})^m \]

- Minimization yields the iteration scheme:
  \[ u_{ih}^{-1} = 1 + \left( \frac{\| \Phi(x_h) - \Phi(v_i) \|^2}{\eta_i} \right)^{1/(m-1)} \]
  \[ v_i = \frac{\sum_{h=1}^{n} (u_{ih})^m K(x_h, v_i)x_h}{\sum_{h=1}^{n} (u_{ih})^m K(x_h, v_i)} \]

- For the Gaussian Kernel:
  \[ u_{ih}^{-1} = 1 + 2\left( \frac{1 - K(x_h, v_i)}{\eta_i} \right)^{1/(m-1)} \]
One Class Support Vector Machines

• The idea is to find the smallest enclosing sphere in kernel space of radius $R$ centered at $v$: $\|\Phi(x_j) - v\|^2 \leq R^2 + \xi_j \quad \forall j \, , \xi_i \geq 0$ , are the slack variables

• The Lagrangian for the above is:
  
  \[ L = R^2 - \sum_j (R^2 + \xi_j - \|\Phi(x_j) - v\|^2)\beta_j - \sum_j \xi_j \mu_j + C \sum_j \xi_j \]

  $, \beta_i \geq 0$ and $\mu_i \geq 0$ are Lagrange multipliers, $C \sum_j \xi_j$ is the penalty term with $C$ -user defined const.

• Taking the derivative wrt $\xi_j$, $R$, $v$ and the KKT complementarity conditions yield the following QP:
  
  $\sum_j \beta_j = 1, \quad v = \sum_j \beta_j \Phi(x_j), \quad \beta_j = C - \mu_j$

  $\xi_j \mu_j = 0, \quad (R^2 + \xi_j - \|\Phi(x_j) - v\|^2)\beta_j = 0$

• $\xi_i > 0$ , for outliers and $\xi_i = 0, 0 < \beta_i < C$ for the support vectors
Example of one class SVMs

One class SVM with a linear kernel applied to a data set with outliers. The gray line shows the projection in input space of the smallest enclosing sphere in feature space.
Extension of one class SVMs to Clustering

• Similar to Kernel SVM but here the SVMs are applied to partition the space.

• The Voronoi regions are now spheres:

\[ \pi_i^\Phi(\rho) = \{ x_j \in \pi_i^\Phi \text{ and } \| \Phi(x_j) - v_i^\Phi \| < \rho \} \]

• Algorithm:
  1. Project the data set \( X \) into a feature space \( \mathcal{F} \), by means of a nonlinear mapping \( \Phi \)
  2. Initialize the codebook \( V^\Phi = (v_1^\Phi, \ldots, v_k^\Phi) \) with \( v_i^\Phi \in \mathcal{F} \)
  3. Compute \( \pi_i^\Phi(\rho) \) for each center \( v_i^\Phi \)
  4. Apply One Class SVM to each \( \pi_i^\Phi(\rho) \) and assign the center obtained to \( v_i^\Phi \)
  5. Go to step 2 until any \( v_i^\Phi \) changes.
  6. Return the feature space codebook.