Unsupervised Clustering Algorithms Comparison for On-line Sensor Clustering

Steven Lowette & Kristof Van Laerhoven

Characteristics overview

Algorithm	Implemented	Online	Learning Kind	Number of Clusters	Visualisation	Network Topology	Speed	Performance on Clustered Data	Performance on Spread Data	Remarks
Self Organising Map	Yes	Yes	Soft competitive	Variable	All dim.	Fixed	++	++	−−	
Recurrent Self Organising Map	Yes	Yes	Soft Competitive	Variable	All dim.	Fixed	+	++	−	The same as the SOM, but with a 'memory' for Temporal Sequence Processing
K-Means Clustering	Yes	Yes	Hard competitive	Fixed	2 dim. only	Variable	+	−	+	Learning rule 1/n gives mean over all input in cluster so far
Sequential Leader Clustering	No	Yes	Hard competitive	Variable	2 dim. only	Variable	+	+/−	+/−	Low accuracy and flexibility, because the clusters are static
Growing K-Means Clustering	Yes	Yes	Hard competitive	Variable	2 dim. only	Variable	+	+	+/−	Because of the movement of the clusters, old inputs can get out of it's cluster's radius
Growing K-means Clustering with Maximum Cluster Size (using FIFO)	Yes	No	Hard competitive	Variable	2 dim. only	Variable	−−	+	+/−	The gain of flexibility on the input data is overshadowed by the loss of on-line processing
Neural Gas	Yes	Yes	Soft competitive	Fixed	2 dim. only	Variable	+	+/−	+	
Neural Gas with Competitive Hebbian Learning	Yes	Yes	Soft competitive	Fixed	2 dim. only	Variable	+	+/−	+	The addition of the edges between the neurones has no influence on the learning
Growing Neural Gas	Yes	Yes	Soft competitive	Variable	2 dim. only	Variable	++	−	+	

Algorithm overview

Self Organizing Map (SOM)

Parameters

- Number of X-Neurons: Horizontal number of neurons.
- Number of Y-Neurons: Vertical number of neurons.
- Metric Function: Function used to calculate the distance between the inputvector and the neurons.
- Neighbor Metric Function: Function used to calculate the distance between the neighboring neurons.

- Neighbor Radius: Value of the excitation radius around the winning neuron.
- Update Function: Function used to calculate the learning rate Eta, depending on the number of times a neuron was triggered.
- Initial Learning Rate: Starting value of the learning rate Eta.
- Torus: Whether or not the 2-dimensional network should be treated as a torus.

Description of the Algorithm

1. `Read new input`

1. `Find winning neuron:`
    - `select distance function to be used`
    - `find neuron with minimal distance to input`

1. `Update neurons. For each neuron and component i:`
    - `select the neighbor distance function`
    - `calculate the neighbor value: NbrVal (centered around winning neuron)`
    - `select the update function for the learning rate`
    - `calculate the learning rate: Eta`
    - `update: NewNeur(i) = OldNeur(i) + NbrVal * Eta * (Input(i)-OldNeur(i))`

Recurrent Self Organising Map (RSOM)

Parameters

- Number of X-Neurons: Horizontal number of neurons.
- Number of Y-Neurons: Vertical number of neurons.
- Metric Function: Function used to calculate the distance between the inputvector and the neurons.
- Neighbor Metric Function: Function used to calculate the distance between the neighboring neurons.
- Neighbor Radius: Value of the excitation radius around the winning neuron.
- Update Function: Function used to calculate the learning rate Eta, depending on the number of times a neuron was triggered.
- Initial Learning Rate: Starting value of the learning rate Eta.
- Alfa: Memory Value indicating the amount of memory of previous steps. If Alfa=1, then the RSOM is equivalent to the SOM.
- Torus: Whether or not the 2-dimensional network should be treated as a torus.

Description of the Algorithm

1. `Read new input`
2. `Update the difference vectors. For each neuron and component i:`
    - `NewDiff(i) = (1 - Alfa) * OldDiff(i) + Alfa * (Input(i) - OldNeur(i))`

1. `Find winning neuron:`
    - `select distance function to be used`
    - `find difference vector with minimal distance to the origin`
    - `the winning neuron is the neuron which corresponds to this winning difference vector`

1. `Update neurons. For each neuron and component i:`
    - `select the neighbor distance function`
    - `calculate the neighbor value: NbrVal (centered around winning neuron)`
    - `select the update function for the learning rate`
    - `calculate the learning rate: Eta`
    - `update: NewNeur(i) = OldNeur(i) + NbrVal * Eta * NewDiff(i)`

K-Means Clustering (KMC)

Parameters

- Number of Clusters.
- Metric Function: Function used to calculate the distance between the inputvector and the clusters.
- Adaptation Rule: Function used to calculate the adaptation value Eta

, depending on the number of inputs in the cluster.

Description of the Algorithm

1. `Read New Input`
2. `Find the winning cluster:`
   - `select distance function to be used`
   - `find cluster with minimal distance to input`

1. `Update:`
   - `select adaptation rule`
   - `calculate adaptation rate: Eta`
   - `update each component i of the winning cluster: NewClus(i) = OldClus(i) + Eta * (Input(i)-OldClus(i))`

Sequential Leader Clustering (SLC)

Parameters

- Metric Function: Function used to calculate the distance between the inputvector and the clusters.
- Threshold: Radius around a cluster's center, in which inputs must fall

Description of the Algorithm

1. `Read New Input`
2. `Find the winning cluster`
   - `select distance function to be used`
   - `find cluster with minimal distance to input`

1. `If distance to winning cluster is under threshold then`
   - `input belongs to winning cluster`

   `Else`
   - `create new cluster = input`
   - `input belongs to new cluster`

Growing K-Means Clustering (GKMC)

Parameters

- Metric Function: Function used to calculate the distance between the inputvector and the clusters.
- Adaptation Rule: Function used to calculate the adaptation value Eta, depending on the number of inputs in the cluster.
- Threshold: Radius around a cluster's center, in which inputs must fall

Description of the Algorithm

1. `Read New Input`
2. `Find the winning cluster:`
   - `select distance function to be used`
   - `find cluster with minimal distance to input`

1. `If distance to winning cluster is under threshold then update:`
   - `select adaptation rule`
   - `calculate adaptation rate: Eta`
   - `update winning cluster: NewClus(i) = OldClus(i) + Eta * (Input(i)-OldClus(i))`

   `Else`
   - `create new cluster = input`

Growing K-means Clustering Algorithm with Maximum Cluster Size, using FIFO (GKMC_FIFO)

Parameters

- Metric Function: Function used to calculate the distance between the inputvector and the clusters.
- Maximum Cluster Size: Maximal number of inputs in one cluster
- Threshold: Radius around a cluster's center, in which inputs must fall

Description of the Algorithm

1. `Read New Input`
2. `Find the winning cluster:`
   - `select distance function to be used`
   - `find cluster with minimal distance to input`

1. `If distance to winning cluster is under threshold then update:`
   - `make input a 'member' of the winning cluster`
   - `update winning cluster: WinClus = Mean(Last NrLast Input in this Cluster)`

   `Else`
   - `create new cluster`
   - `make input 'member' of new cluster`

Neural Gas Algorithm (NGA)

Parameters

- Number of Neurons.
- Metric Function: Function used to calculate the distance between the inputvector and the clusters.
- Maximum Position: see Final Epsilon and Final Lambda.
- Initial Epsilon: Global learning rate at position 0.
- Final Epsilon: Global learning rate at the Maximum Position.
- Initial Lambda: Sequence learning rate at position 0 (see also Description of the Algorithm).
- Final Lambda: Sequence learning rate at the Maximum Position (see also Description of the Algorithm).

Description of the Algorithm

1. `Read New Input`
2. `Calculate all the distances of the (fixed number of) clusters to the input`
3. `Make a list of the increasing distances and their corresponding clusternumber: Sequence()`
4. `Update the clusters. For each cluster w and each component i:`
   - `calculate current learning rate: EpsCur = Epsi * (Epsf/Epsi)^(Pos/MaxPos)`
   - `calculate current distance function:`

     `DistCur = Exp(-Sequence(w) / (Lmbi*(Lmbf/Lmbi)^(Pos/MaxPos)))`
   - `update: NewClus = OldClus + EpsCur * DistCur * (Input(i)-OldClus(i))`

Neural Gas Algorithm with Competitive Hebbian Learning (NGA_CHL)

Parameters

- Number of Neurons.
- Metric Function: Function used to calculate the distance between the inputvector and the clusters.
- Maximum Position: see Final Epsilon and Final Lambda.
- Initial Epsilon: Global learning rate at position 0.
- Final Epsilon: Global learning rate at the Maximum Position.
- Initial Lambda: Sequence learning rate at position 0 (see also Description of the Algorithm).
- Final Lambda: Sequence learning rate at the Maximum Position (see also Description of the Algorithm).
- Initial Maximum Edge Age: Maximal age for a non-refreshed edge at position 0.
- Final Maximum Edge Age: Maximal age for a non-refreshed edge at the Maximum Position.

Description of the Algorithm

1. `Read New Input`
2. `Calculate distances`
   - `select distance function to be used`
   - `calculate all the distances of the (fixed number of) clusters to the input`

1. `Make an increasing list of the distances to the input and their corresponding clusternumber: Sequence()`
2. `Update the clusters. For each cluster w and each component i:`
   - `calculate current learning rate: EpsCur = Epsi * (Epsf/Epsi)^(Pos/MaxPos)`
   - `calculate current distance function:`

```
DistCur = Exp(-Sequence(w) / (Lmbi*(Lmbf/Lmbi)^(Pos/MaxPos)))
```
- `update: NewClus = OldClus + EpsCur * DistCur * (Input(i)-OldClus(i))`

1. `Maintain edges:`
    - `detect two clusters nearest to input: Winner and Winner2`
    - `'refresh' the edge between Winner and Winner2: set age=0 or create`
    - `increment all edges' age`
    - `for every edge: if age exceeds MaxAge then remove edge`

Growing Neural Gas (GNG)

Parameters

- Metric Function: Function used to calculate the distance between the inputvector and the clusters.
- Maximal Edge Age: Maximal age for a non-refreshed edge.
- New Cluster Threshold: Number of steps after which a new cluster is to be created.
- Winning Learning Rate: Learning Rate for the winner.
- Neighbor Learning Rate: Learning rate for the clusters directly connected to the winning cluster with edges.
- Alfa: Fraction by which the local errors of the winning and the second best clusters are decreased, when creating a new cluster.
- Beta: Fraction by which all the clusters' local errors are decreased each step.

Description of the Algorithm

1. `Read New Input`
2. `Calculate distances:`
    - `select distance function to be used`
    - `find two clusters nearest to input: Winner and Winner2`

1. `Add squared distance between input and Winner to winner's 'error'`
2. `'Refresh' the edge between Winner and Winner2: set age=0 or create`
3. `Adapt Clusters:`
    - `Adapt Winner: Winner(i) = Winner(i) + EpsWin * (Input(i) - Winner(i))`
    - `Adapt Clusters connected to winner through edge:`

      `Nbr(i) = Nbr(i) + EpsNbr * (Input(i) - Nbr(i))`

1. `Maintain edges:`
    - `increment all edges' age`
    - `for every edge: if age exceeds MaxAge then remove edge`

1. `Delete Clusters who are not connected any more`
2. `If Number of steps is the integer multiple of chosen Lambda Then`
    - `find cluster with biggest 'error': Winner`
    - `find neighboring cluster with second-biggest 'error': Winner2`
    - `decrease 'errors' of Winner and Winner2 by factor Alfa`
    - `create new cluster and interpolate place and 'error' between Winner and Winner2`
    - `connect new cluster to Winner and Winner2`
    - `delete edge between Winner and Winner2`

1. `Decrease 'errors' of all clusters by factor Beta`

Definitions

Metrics

- City Block Metric: $d(\vec{a},\vec{b}) = \sum_{i=1}^{n} |a_i - b_i|$
- Euclidean Metric: $d(\vec{a},\vec{b}) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$
- Maximum Metric: $d(\vec{a},\vec{b}) = \max_i |a_i - b_i|$

Neighbor Functions

- Triangular: $NeighFactor(\vec{a}) = \begin{cases} d(\vec{a},Winner) < NeighRadius: & \dfrac{NeighRadius - d(\vec{a},Winner)}{NeighRadius} \\ d(\vec{a},Winner) > NeighRadius: & 0 \end{cases}$
- Gaussian: $NeighFactor(\vec{a}) = e^{-\left(\frac{d(\vec{a},Winner)}{NeighRadius}\right)^2}$
- Excitation/Inhibition: $NeighFactor(\vec{a}) = \left[1 - \left(\dfrac{d(\vec{a},Winner)}{NeighRadius}\right)^2\right] e^{-\left(\frac{d(\vec{a},Winner)}{NeighRadius}\right)^2}$

  Remark: If $NeighFactor < 0$

, the adaptation is slightly modified to avoid overflow, by keeping the vector components between 0 and 255, as it should be.

Adaptation Rules

- Constant Learning Rate: $EtaFactor = Etai$,

  where $Etai$ is the Initial Learning Rate.
- Linear decreasing Learning Rate: $EtaFactor = \dfrac{Etai}{NrElements}$,

  where $NrElements$ is the number of elements associated with the current cluster or the number of times the current cluster has been triggered.
- Square Root Learning Rule: $EtaFactor = \dfrac{Etai}{\sqrt{NrElements}}$
- Exponential Learning Rule: $EtaFactor = \dfrac{Etai}{e^{NrElements}}$