Possibilistic C-Means Clustering ToolBox

Hung Tran-Nam

Abstract

Keywords:

Contents

1. PCM_()

The `PCM_` function performs Possibilistic C-Means (PCM) clustering on the input data.

Definition

Possibilistic C-Means (PCM) is a clustering algorithm that extends Fuzzy C-Means (FCM) to handle uncertain memberships and noisy data points. It allows each data point to belong to multiple clusters with varying degrees of membership.

Syntax

```
results = PCM_(Data, param, varargin)
```

Inputs

- `Data` - A matrix where each column represents a data point.
- `param` - A structure containing the following fields:
  - `kClust` (required) - Number of clusters.
  - `maxIter` (optional) - Maximum number of iterations (default: 100).
  - `mFuzzy` (optional) - Fuzziness parameter (default: 2.0).
  - `epsilon` (optional) - Convergence threshold (default: 1e-5).
  - `alphaCut` (optional) - Threshold for noise identification (default: 0.5).
  - `K` (optional) - Scaling factor for eta calculation (default: 0.5).

  - `x` (optional) - Support points for the PDFs.

- `varargin` - Optional parameters for visualization:
  - `'Visualize'` - Visualization type: `'None'`, `'CDF'`, or `'CDE'` (default: `'None'`).

Outputs

- `results` - A structure containing clustering results:
  - `Cluster.U` - Final partition matrix.
  - `Data.fv` - Representative PDFs of clusters.
  - `iter` - Number of iterations performed.
  - `ObjFun` - Final value of the objective function.
  - `Data.Data` - Input data.
  - `Cluster.IDX` - Cluster indices for each data point.
  - `Dist.D` - Distance matrix between representative PDFs and data.
  - `isnoise` - Data points identified as noise.

Algorithm Steps

1. **Initialization**: Initialize cluster centers and membership values.
2. **Membership Update**: Update memberships based on distance to cluster centers and fuzziness parameter.
3. **Cluster Center Update**: Update cluster centers based on new memberships.
4. **Convergence Check**: Check convergence based on objective function change or maximum iterations.
5. **Noise Identification**: Identify noise data points based on alpha cut threshold.
6. **Output**: Return final cluster assignments, representative PDFs, and other diagnostic information.

Example

Consider a dataset where each data point represents measurements in a two-dimensional space. We apply PCM clustering to this dataset with the following parameters:

```
% Define the input data
x =

Data = normpdf(x, mu, sigma)

% Define the parameters
param.kClust = 3;        % Number of clusters
param.maxIter = 100;     % Maximum number of iterations
param.mFuzzy = 2.0;      % Fuzziness parameter
param.epsilon = 1e-5;    % Convergence threshold
param.alphaCut = 0.5;    % Threshold for noise identification
param.K = 0.5;           % Scaling factor for eta calculation
param.x = linspace(0, 1, 100); % Support points for the PDFs

```

```matlab
% Call the PCM_ function
results = PCM_(Data, param, 'Visualize', 'None');

% Display the results
disp('Cluster Indices:');
disp(results.Cluster.IDX);

disp('Representative PDFs:');
disp(results.Data.fv);

disp('Number of Iterations:');
disp(results.iter);

disp('Objective Function Value:');
disp(results.ObjFun);
```

In this example, we generate random data points in a two-dimensional space and apply PCM clustering to identify clusters in the data. The `PCM_` function computes cluster memberships, updates cluster centers, and identifies noise points based on specified parameters.

2. ExtractKernel()

The MATLAB function `ExtractKernel` computes the kernel density estimate (pdf) for images in an `ImageDatastore`.

Description

`ExtractKernel` filters grayscale images from the `ImageDatastore imds`, computes the bandwidth (h) for kernel density estimation, and returns the pdf values and corresponding x values.

Syntax

```matlab
[pdf, x] = ExtractKernel(imds)
[pdf, x] = ExtractKernel(imds, Name, Value)
```

Input Arguments

- `imds` - A `matlab.io.datastore.ImageDatastore` object containing images.

Optional Name-Value Pair Arguments

- `'numPoints'` - Number of points for kernel density estimation (default: 1000).
- `'extensions'` - File extensions of images in `imds` (default: {'.png'}).

Output Arguments

- `pdf` - Kernel density estimate values for each image.
- `x` - Points at which the kernel density estimate is evaluated.

Example

```
folderPath = 'link/to/folder/data'
imds = imageDatastore(folderPath);
[pdf, x] = ExtractKernel(imds, 'numPoints', 500);
```

See also

- `ksdensity`
- `imageDatastore`

3. PlotPDFeachIteration()

The `PlotPDFeachIteration` function plots the probability density functions (PDFs) from a data matrix.

Description

The `PlotPDFeachIteration` function takes in a data matrix where each column represents a PDF and a vector of labels corresponding to each PDF. An optional x-axis vector can also be provided. The function plots each PDF with a color corresponding to its label, adds a legend, axis labels, and a title.

Syntax

```
h = PlotPDFeachIteration(data, labels, x)
```

Inputs

- `data` - A matrix containing PDF data. Each column represents a PDF.
- `labels` - A vector containing the labels for each PDF.
- `x` (optional) - A vector of x-axis values. If not provided, a default vector is created.

Outputs

- `h` - Handle to the created figure.

Example

The following example demonstrates how to use the `PlotPDFeachIteration` function to plot PDFs from a dataset.

```
% Define the input data
data = rand(500, 3); % 3 random PDFs with 500 points each
labels = [1, 2, 3];  % Labels for the PDFs
x = linspace(-0.5, 1.5, 500); % X-axis values

% Call the PlotPDFeachIteration function
h = PlotPDFeachIteration(data, labels, x);
```

In this example, the input data is randomly generated. The function is called with the data, labels, and x-axis values to plot the PDFs.

4. validityClustering()

The `validityClustering` function validates the clustering results using various indices.

Description

The `validityClustering` function calculates various indices to validate the clustering results. These indices include the partition coefficient (PC), classification entropy (CE), Xie and Beni's index (XB), silhouette coefficient (SC), Rand Index (RI), Adjusted Rand Index (ARI), and G-mean.

Syntax

```
results = validityClustering(results, param)
```

Inputs

- `results` - A structure containing the clustering results and related data.
- `param` - A structure containing parameters for validation.

Outputs

- `results` - A structure containing the original clustering results and the added validation indices.

Example

The following example demonstrates how to use the `validityClustering` function to validate clustering results.

```
% Define the input results and param structures
results.Data.Data = rand(2, 100); % Random data
results.Cluster.U = rand(3, 100); % Random membership matrix
results.Cluster.IDX = randi([1, 3], 1, 100); % Random cluster indices
results.Dist.D = rand(3, 100); % Random distance matrix

param.val = 3;
param.mFuzzy = 2;
param.truelabels = randi([1, 3], 1, 100); % Random true labels

% Call the validityClustering function
results = validityClustering(results, param);
```

In this example, the input data is randomly generated. The function is called with the results and param structures to validate the clustering results.