Tutorial on How to use Spectral Clustering Algorithm for Creating Regions from Geospatial Data

Farzan Masrour

January 13, 2017

This document describes how to use the Spectral Clustering method described by Cheruvelil et al.This tutorial uses the program R.

1 preparation

Before you can use this code you need to install the following packages: Matrix, geigen, rARPACK, maps, WDI, RColorBrewer, and maptools. Note that the R packages maps, WDI, RColorBrewer, maptools are only required for plotting the clustering results. To install these packages, use the following commands in R.

```
> install.packages("Matrix")
> install.packages("geigen")
> install.packages("rARPACK")
> install.packages("maps")
> install.packages("WDI")
> install.packages("RColorBrewer")
> install.packages("maptools")
```

Although you only have to install the packages once, you will need to load them every time you want to use this code. Use the following code to load the packages.[1]

```
> # First Load the necessary packages
> library("Matrix")
> library("geigen")
> library("rARPACK")
> library(maps)
> library(WDI)
> library(RColorBrewer)
> library("maptools")
```

[1] The file "example.R" contains the code in this document.

The next step is setting your working directory to be the folder that contains the code and data. In order to use the functions and methods in this code you also need to call main.R file using source() command as follow.

```
> # Second generate data objects and load spectral clustering code
> source("GenerateHU12.R")
> source("main.R")
```

2 Spectral Clustering

In this section, we describe how to do spectral clustering without deep understanding of the algorithm.

2.1 Generate the data

Like any other algorithm, spectral clustering needs some variable as input. Which includes:

- **data**: An n by p matrix where n is number of observation(e.g., HU12's) and p is the number of measurement for each observation(e.g., natural geographical variables for each HU12).
- **cluster.number:** Number of clusters.
- **conMatrix**: Constraint matrix or contiguity matrix.
- **conFactor**: Contiguity constraint factor.

We can read the HU12 data by loading **HU12SpectralClustering.RData** file. This file contains dataTerr, dataFW, and dataTerrFw as three option of data, latLong18876 which is the location of each observation on Map, and NB18876 the contiguity constraint data frame that can be used to build the conMatrix.

```
> #Loading Data
> load("HU12SpectralClustering.RData")
```

After loading the Hu12SpectralClustering.RData, you can generate the input data for spectral clustering algorithm by calling generateData() function[2]. For the sake of clarity we do a toy example in this document. Let consider we are interseted in Missouri state dataTerrFW's data.

```
> # generate the input when we restrict the
> # data to dataTerrFW's of state Missouri,
> # and no islands inculded.
> input <- generateData(type= dataTerrFW, islandsIn = F,
+                       states = c("MO"),conFactor=1)
```

In the above example we stored the output of generateData() in a variable called input. Know we are ready to run the spectral clustering methods.

[2]For description of functions look at section 3.1 in this document

2.2 Clustering

By using this code you have three options for doing the clustering. First, using one-step **speCluster()** function which is easy and simple but slow. Second, using two-step approach by calling **stepOne()** and **stepTwo()** functions that give you the ability to check different size of clusters in shorter time. Finally, **step by step** approach, which is more complicated and needs some knowledge of Spectral clustering algorithm steps.To understand the difference between these three approaches we need to understand the spectral clustering algorithm.

We can divide the algorithm to three parts as follow:

Algorithm steps:

1. **Preprocess**
   Input: data, conMatrix
   (a) Detecting the outliers and replace them with mean of data
   (b) Using the principal component to reduce the dimension of parameters space.

   Output: ConMatrix, outlier, dataAfterPC

2. **Main algorithm**
   Input: ConMatrix, outlier, dataAfterPC, cluster.number(k)
   (a) Compute similarity matrix

$$S_{i,j} = exp(\frac{dist(x_i, x_j)}{2\sigma^2})$$

   Where x_i and x_j are row i and j of the data matrix, and σ is median of pairwise distance.Then

$$S = S \circ S_c$$

   where S_c is contiguity matrix

   (b) compute Laplacian matrix .
      i. Diagonal matrix: $d_{i,i} = \sum_{j=1}^{n} S_{i,j}$
      ii. Compute Laplacian matrix: $L = D^{-1}S$

   (c) Find u_1, u_2, ...,u_k, the k top eigenvectors of L, where k is the clustering.number. Form matrix

$$U = [u_1 \dots u_k]$$

   (d) clusters U in to k cluster using kmean algorithm.
   (e) assign the original point x_i to cluster j if and only if row i of matrix U was assigned to cluster j.

   Output: clusters

3. **postprocess**

   Input: clusters, latLon, data

   (a) error computation

      i. compute Sum of Squired Between clusters SSB

      ii. compute Sum of Squired within clusters SSW

   (b) Mapping

   Output: SSB, SSW, graphs

Now we can explain the three approachs we introduced above. Lets get back to our toy example.

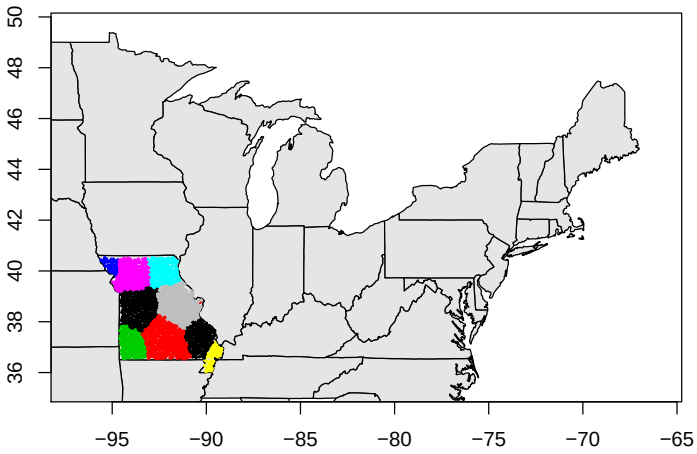
```
> input <- generateData(type= dataTerrFW, islandsIn = F,
+                       states = c("MO"),conFactor=1)
```

Approach 1. speCluster()

- **1.a ... 3.a** Use speCluster() function.

- **3.b** plot the results using mapping function

```
> # 1.a ... 3.a using speCluster and stor the result in the results variable
> results <- speCluster(data= input$data, conMatrix=
+                           input$conMatrix, cluster.number=10)
> summary(results)

         Length Class  Mode
clusters 1748   -none- numeric
SS          2   -none- list

> results$SS

$SSW
[1] 47925.58

$SSB
[1] 17244.8

> head(results$clusters)

1 2 3 4 5 6
3 3 3 3 3 3

> mapping(lat = input$latLong[,1],long=input$latLong[,2],
+          clusters= results$clusters)
```



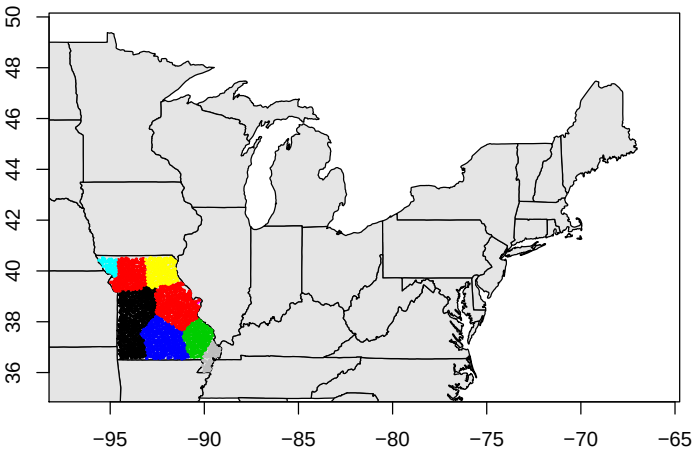
If you look at the steps 2.c and 2.d of the above algorithm, you can notice that the column size of U in 2.c is equal to the number of clusters in 2.d. For example if we want 10 clusters we only need to calculate first 10 eigenvecotrs and build U with 10 columns. But what if we want to check 20 clusters? We need to call the speClus function again from beginning. However if we calculated U with 20 columns then we could check all the clustering options less than 20 without going back and recalculate all the steps from beginning. This is the main idea behind the second approach.

Approach 2. Two step

- **1.a...2.C** using stepOne() function.
- **2.d... 3.a** using stepTwo() function.
- **3.b** plot the results using mapping function

```
> # example.2. Two steps
> results1 <- stepOne(input$data, conMatrix= input$conMatrix,ncol=20)
> summary(results1)

            Length Class  Mode
dataAfterPC 33212  -none- numeric
U           34960  -none- numeric
```

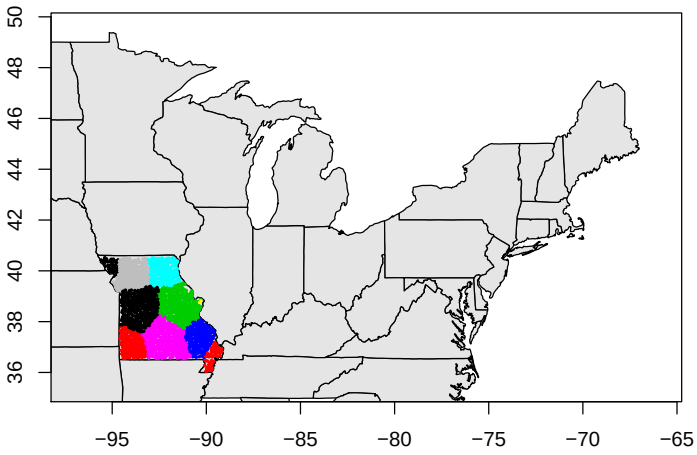
```
> results2 <- stepTwo(data= results1$dataAfterPC, U = results1$U,
+                     cluster.number=10)
> summary(results2)

         Length Class  Mode
clusters 1748   -none- numeric
SS          2   -none- list

> results2$SS

$SSW
[1] 47925.58

$SSB
[1] 17244.8

> mapping( lat = input$latLong[,1], long = input$latLong[,2],
+          clusters= results2$clusters)
>
> # since the U matrix has 20 column we can to clustering up
> # to 20 clusters using output of stepOne function
>
> # these inputs (dataAfterPC and U) have not been generated yet - jsta
> # results22 <-  stepTwo(data= input$dataAfterPC, U = input$U,
> #                       cluster.number=20)
> # results22$SS
> # mapping( lat = input$latLong[,1], long = input$latLong[,2],
> #          clusters= results22$clusters)
>
```



Approach 3.Step by Step

- **1.a** Find the outlier using outlierDetector() function.
- **1.b** Reduce the data dimension using prinComp() function.
- **2.a** Compute similarity matrix using similarity() function.
- **2.b and 2.c** Compute Laplacian and then U matrix using produceU() function.
- **2.d and 2.e** Calculate the clusters using kmeansU() function.
- **3.a** Evaluate the between and within sum squired error using sumSquares() function.
- **3.b** plot the results using mapping() function.

```
> #Preprocess
>
> outId <-outlierDetector( input$data )
> dataAfterPC <- prinComp(data = input$data, outId = outId)
> ###############################################
> # Spectral clustering Algorithm
> similarity <- similarity(data = dataAfterPC,
+                          neighbors=input$conMatrix)
> U <- produceU( similarity= similarity, ncol=20)
> clusters <- kmeansU(data=U, cluster.number = 10)
> ###############################################
> # Results
> SS <- sumSquares(data=dataAfterPC, clusters= clusters)
> SS

$SSW
[1] 47925.58

$SSB
[1] 17244.8

> mapping( lat = input$latLong[,1],long=input$latLong[,2], clusters= clusters)
>
```



3 Functions, Variables and code Structure

In this section, we provide more details of all the functions and the variables in this code. Also you can find a short description of the structure of the code that might be helpful in understanding how to use this code.

3.1 Functions

---

generateData()

Description
Generate the data for clustering.

Usage

generateData(type,...)

```
>   #Default method:
>   generateData(type, islandsIn =F , states= vectors(),
+                conFactor=1 )
```

Arguments

- type: three options dataTerr, dataFW, and dataTerrFW
- islandIn: if T the islands will be included
- states: a vector of states names that have to be included
- conFactor: contiguity constraint factor

Returns
a list with three elements: data, conMatrix, and latLong

---

kmeansU()

Description
Perform k-means clustering on the U matrix.

Usage

kmeansU(data,...)

```
>   #Default method:
>   kmeansU(data , cluster.number,
+           repetition = 400, iter.max = 400 )
```

Arguments

- data: numeric matrix U
- cluster.number: The number of clusters.
- iter.max: The maximum number of iterations allowed
- repetition: How many random sets should be chosen for as the initial centers

Returns
cluster: A vector of integers(from 1:cluster.number) indicating the cluster to which each point is allocated

---

mapping()

Description
Using map database to draw clusters on the US Map. **Usage**

```
>   #Default method:
>   mapping(long, lat, clusters)
```

Arguments

- long: A numeric vector of longitude location of the points.
- lat: A numeric vector of latitude location of the points.
- clusters: The vector of integers indicating the cluster to which each point is allocated.

---

neighborMatrix()

Description
Compute contiguity Matrix **Usage**

```
>   #Default method:
>   neighborMatrix(NB,conFactor=1)
```

Arguments

- NB: The contiguity constraint data frame
- conFactor: contiguity constraint factor

Returns
conMatrix: Contiguity Matrix

---

outlierDetector()

Description
Compute the outlier of the data using Principal component.
Usage

```
>   #Default method:
>   outlierDetector(data, outlier.Threshold = 0.2 )
```

Arguments

- data: a numeric data frame or matrix
- outlier.Threshold: The Threshold which makes a data outlier.

Returns
outId: A logical vector which specifies all the outliers.

---

prinComp()

Description
Run the principal component algorithm on the data to reduce dimension **Usage**

```
>   #Default method:
>   prinComp(data, outId, showPC = F)
```

Arguments

- data: a numeric data frame or matrix
- outId: A logical vector which specifies all the outliers.
- showPC: A logical value indicating whether principal component should be return or not.

Returns
In case showPC is FALSE:
dataNew: After Principal component data
In case showPC is TRUE:
PC: returns a list with class "prcomp", for more information look at prcomp function of R.

---

produceU()

Description
Given n by n similarity this function first calculates the Laplacian matrix L

$$d_{i,i} = \sum_{j=1}^{n} S_{i,j}$$

$$L = D^{-1}S$$

Then generates n by 'ncol' matrix U of top 'ncol' eigenvectors of L.

Usage

produceU(similarity, ncol ,...)

```
> #Default method:
> produceU(similarity, ncol , type=2, all.eig = F)
```

Arguments

- similarity: an n by n matrix.
- ncol: number of columns of the output matrix U.
- type: The algorithm that should be choose,Options are 1, 2, and 3.
- all.eig: a logical value indicating whether all the eigenvector should be compute or not.

Returns
U: n by 'ncol' numeric matrix that contains the 'ncol' top eigenvectors of Laplacian matrix as column

similarity()

Description
Compute similarity matrix. First $S_{i,j} = exp(\frac{dist(x_i,x_j)}{2\sigma^2})$ Where x_i and x_j are row i and j of the data matrix, and σ is median of pairwise distance.Then

$$S = S \circ S_c$$

where S_c is contiguity matrix. returns S.

Usage

similarity(data, neighbors)

Arguments

- data: n by p numeric matrix or data frame.
- neighbors: a square numeric matrix which specifies contiguity matrix.

Returns
An n by n numeric matrix, that element[i,j] is the similarity index of observation i and observation j.

---

speCluster()

Description
Perform Spectral Clustering on a data matrix.
Usage

speCluster(data,...)

```
>   #Default method:
>  speCluster(data, conMatrix, cluster.number,
+             iter.max=400, repetition= 400 )
```

Arguments

- data: A numeric data frame or matrix.
- conMatrix: Contiguity matrix.
- cluster.number: The number of clusters.
- iter.max: The maximum number of iterations allowed for kmeans step.
- repetition: How many random sets should be chosen for as the initial centers in kmeans step.

Returns
A list contains two parts:

- clusters: A vector of integers(from 1:cluster.number) indicating the cluster to which each point is allocated.
- SS: A list with two values SSW for Sum Squared Within and SSB for Sum Squared Between

---

stepOne()

Description
This function Computes the data after Principal component.
Usage

stepOne(data,...)

```
>   #Default method:
>   stepOne(data, conMatrix, ncol)
```

Arguments

- data: A numeric data frame or matrix.
- conMatrix: Contiguity matrix.
- ncol: number of columns of the output matrix U

Returns

A list contains two parts:

- dataAfterPC: After Principal component data
- U: n by ncol numeric matrix that contains the ncol tops eigenvectors of Laplacian matrix as column.

---

stepTwo()

Description
Perform Spectral Clustering on U matrix.

Usage

stepTwo(data,...)

```
>   #Default method:
>   stepTwo(data, U, cluster.number= cluster.number,
+           iter.max=400, repetition=400)
```

Arguments

- data: A numeric data frame or matrix.
- U: A numeric matrix
- cluster.number: The number of clusters.
- iter.max: The maximum number of iterations allowed for the kmeans step.
- repetition: How many random sets should be chosen for as the initial centers in kmeans step.

Returns
A list contains two parts:

- clusters: A vector of integers(from 1:cluster.number) indicating the cluster to which each point is allocated.

- SS: A list with two values SSW for Sum Squared Within and SSB for Sum Squared Between

---

sumSquares()

Description
Given the data and clusters vector this function computes the between and within sum squared errors.

Usage

```
>   #Default method:
>   sumSquares(data, clusters)
```

Arguments

- data: After Principal component data
- clusters: The vector of integers indicating the cluster to which each point is allocated.

Returns

A list with two values SSW for Sum Squared Within and SSB for Sum Squared Between.

3.2 code Structure

The code contains four files these files are designed base on the main steps of algorithm.

- **"Preprocess.R"** This file contains all the functions which are used in the preprocess part:
  1. neighborMatrix()
  2. outlierDetector()
  3. prinComp()
- **"SpectralClustering.R"** This file contains all the functions that are used in the main part of spectral clustering algorithm
  1. similarity()
  2. produceU()
  3. kmeansU()
- **"Postprocess.R"** This file contains all the functions that are used for after clustering process
  1. sumSquares()
  2. mapping()
- **"main.R"** This is the main file of this code which should be called for using the code functions, the following functions are in this file
  1. speCluster()
  2. stepOne()
  3. stepTwo()
- **"GenerateHU12.R"** The code for generating HU12SpectralClustering.RData file. It also include the generateData() function.