Cluster analysis for spatial data mining and visualisation

Introduction:

Spatial data mining is a process to analyse large geographical databases and extract implicit information from a spatial dataset. It can be viewed as the search for interesting, useful and unexpected, but implicit spatial patterns. Through this process we explore hidden spatial patterns within data.

How spatial data are distributed across a region?
Are they located randomly?
Are they clustered?
Can you see spatial dependency within data?

These are typical questions that can be answered through the process of spatial data analysis. Visualizing the spatial patterns provides better understanding of hidden spatial autocorrelation within data.

Acknowledgements:

Sections of this tutorial (including code) have been derived and or adapted from various public resources and have been duly referenced. Please refer to the reference list at the conclusion of this document for links to the original document created by the copyright holder.

1. Data Preparation:

In this tutorial we will examine different applications of spatial analysis and visualisation.

The first experiment is run on a crime dataset. Exercise preparation and code sourced from material by Veronesi (2015a).

To start we need the following five packages:
sp, raster, spatstat, maptools and *plotrix.*

The dataset provided by the British Police is in *csv* format. You can download all the crime data from the months of May and June 2014 for the whole Britain, but the focus is on the Greater London region, since here the most crimes are committed. You can directly apply **read.csv** function to download the data from a URL:

```
data <- read.csv("http://www.fabioveronesi.net/Blog/2014-05-metropolitan-street.csv")
```

Look at the structure of the dataset simply by using the function 'str':

```
str(data)
```

This dataset provides information about crime: locations, type and the court outcome (if available). For some incidents the location information (coordinates) are not provided, therefore before we can proceed we need to remove NAs from dataset:

```
data <- data[!is.na(data$Longitude)&!is.na(data$Latitude),]
```

How many records have been eliminated?

2. Analysing point patterns

Point pattern analysis is the study of the spatial arrangements of points in (usually 2-dimensional) space. The easiest way to visualize a 2-D point pattern is a map of the locations. For point pattern analysis, the R package **spatstat** will be used.

The crime dataset contains lots of duplicated locations. We can check this by first transforming the dataset into a *SpatialObject*:

```
coordinates(data)=~Longitude+Latitude
```

Then use the function **zerodist** to check for duplicated locations:

```
zero <- zerodist(data)
length(unique(zero[,1]))
```

How many duplicates can you see with in data?

Duplicates are not always due to error. In this situation, they may be simply because more than one person was involved in the crime, in other cases there may be two different crimes for the same location or maybe the crime belongs to several categories.

Regardless of the reason, in **spatstat** the function **remove.duplicates** will remove all duplicates. This sort of problems is often encountered when working with real datasets.

We will now focus on the crimes committed in Greater London region. **Natural Earth** provides this sort of data, so you can download it and import it in R using the following lines:

```
download.file("http://www.naturalearthdata.com/http//www.naturalearthdata.com/download/10m/cultural/ne_10m_admin_1_states_provinces.zip",destfile="ne_10m_admin_1_states_provinces.zip")

unzip("ne_10m_admin_1_states_provinces.zip",exdir="NaturalEarth")

border <- shapefile("NaturalEarth/ne_10m_admin_1_states_provinces.shp")
```

These lines download the shapefile to a compressed archive (.zip). Uncompress this zip to a new folder named *NaturalEarth* in the working directory and open it.

To extract only the border of the Greater London regions we can simply create a subset of the *SpatialPolygons* object as follows:

```
GreaterLondon <- border[paste(border$region)=="Greater London",]
```

Now we need to overlay it with the crime data and eliminate all the points that do not belong to the Greater London region. To do that use the following:

```
projection(data)=projection(border)
overlay <- over(data,GreaterLondon)
data$over <- overlay$OBJECTID_1
data.London <- data[!is.na(data$over),]
```

The first line assigns to the object *data* to the same projection as the object *border*, we can do this safely because we know that the crime dataset is in geographical coordinates (WGS84), the same as *border*.

Then we can use the function over to overlay the two objects. At this point we need a way to extract from *data* only the points that belong to the Greater London region, to do that create a new column and assign to it the values of the overlay object (here the column of the overlay object does not really matter, since we only need it to identify locations where this has some data in it).

In locations where the data is outside the area defined by *border*, the new column will have values of NA, so we can use this information to extract the locations we need with the last line.

We can create a very simple plot of the final dataset and save it in a jpeg using the following code:

```
jpeg("PP_plot.jpg",2500,2000,res=300)
plot(data.London,pch="+",cex=0.5,main="",col=data.London$Crime.type)
plot(GreaterLondon,add=T)
legend(x=-
0.53,y=51.41,pch="+",col=unique(data.London$Crime.type),legend=unique(data.London$Crime.type),
cex=0.4)
```

Now we have a dataset of crimes only for Greater London and we can start our analysis.

2.1 Descriptive Statistics

Point pattern analysis can be seen as an attempt to analyse the occurrence of points in a particular space.

Often the first question asked is simply, *how many points are there?* For example*, how many crimes are committed in a neighbourhood of a city? How does that compare to a differing neighbourhood, or that city as a whole?*

Simple descriptive statistics such as Count, Mean, Median, and Standard Deviation can answer these questions. Applying these, we can describe how dense a pattern is, where the centre of a set of points is, and how dispersed these points are.

Frequency and Density:

Frequency is the most basic way of evaluating a spatial pattern and is simply counting the number of points in your study area. To get density, you divide this total by unit of area or time in whatever units you deem are most appropriate: Hospitals per square mile, violent crimes per month, etc. Frequency and density should almost always be determined at the start of your analysis

We can use mean and standard deviation in 2D space to achieve this. Instead of computing the mean we compute the **mean centre**, which is basically the point identified by the mean value of longitude and the mean value of latitude:

$$C = (\bar{x}, \bar{y}) = \left(\frac{\sum_{i=1}^{n} x_i}{n}, \frac{\sum_{i=1}^{n} y_i}{n} \right)$$

We can compute the standard deviation of longitude and latitude, and the **standard distance**, which measures the standard deviation of the distance of each point from the mean centre. This is important because it gives a measure of spread in the 2D space, and can be computed with the following equation from Wu (2006):

$$d = \sqrt{\frac{\sum_{i=1}^{n} [(x_i - \bar{x})^2 + (y_i - \bar{y})^2]}{n}}$$

In R we can calculate all these indexes with the following simple code:

```
mean_centerX <- mean(data.London@coords[,1])
mean_centerY <- mean(data.London@coords[,2])
standard_deviationX <- sd(data.London@coords[,1])
standard_deviationY <- sd(data.London@coords[,2])
standard_distance <- sqrt(sum(((data.London@coords[,1]-
mean_centerX)^2+(data.London@coords[,2]-mean_centerY)^2))/(nrow(data.London)))
```

We can use the standard distance to have a visual feeling of the spread of our data around their mean centre. We can use the function **draw.circle** in the package **plotrix** to do that:

```
jpeg("PP_Circle.jpeg",2500,2000,res=300)
plot(data.London,pch="+",cex=0.5,main="")
plot(GreaterLondon,add=T)
points(mean_centerX,mean_centerY,col="red",pch=16)
draw.circle(mean_centerX,mean_centerY,radius=standard_distance,border="red",lwd=2)
```

Standard Distance Deviation: is the standard deviation of the distance of each point from the mean center. It is the spatial equivalent of standard deviation and likewise provides one with an idea of the deviation or variance of a dataset as a whole, or the deviation of a particular point from the rest. Standard Deviational Ellipse is a modified version of standard distance that captures the shape of this distribution by showing any directional bias in the pattern.

The problem with the standard distance is that it averages the standard deviation of the distances for both coordinates, so it does not take into account possible differences between the two dimensions. We can take those into account by plotting an ellipse, instead of a circle, with the two axis equal to

the standard deviations of longitude and latitude. We can use again the package **plotrix**, but with the function **draw.ellipse** to do the job:

```
jpeg("PP_Ellipse.jpeg",2500,2000,res=300)
plot(data.London,pch="+",cex=0.5,main="")
plot(GreaterLondon,add=T)
points(mean_centerX,mean_centerY,col="red",pch=16)
draw.ellipse(mean_centerX,mean_centerY,a=standard_deviationX,b=standard_deviationY,border="red",lwd=2)
```

2.2. Complete spatial randomness

Assessing if a point pattern is random is a crucial step of the analysis. If we determine that the pattern is random it means that each point is independent from each other and from any other factor. Complete spatial randomness implies that events from the point process are equally as likely to occur in every regions of the study window. In other words, the location of one point does not affect the probability of another being observed nearby, each point is therefore completely independent from the others (Bivand et al., 2008).

If a point pattern is not random it can be classified in two other ways: clustered or regular. Clustered means that there are areas where the number of events is higher than average, regular means that basically each subarea has the same number of events.

In **spatstat** we can determine which distribution our data have using the G function, which computes the distribution of the distances between each event and its nearest neighbour (Bivand et al., 2008). Based on the curve generated by the G function we can determine the distribution of our data.

Before computing G Function we need to prepare data. In this experiment we can only focus on drug related crime. We can subset the data and remove duplicates as follows:

```
Drugs <- data.London[data.London$Crime.type==unique(data.London$Crime.type)[3],]
Drugs <- remove.duplicates(Drugs)
```

We obtain a dataset with 2745 events all over Greater London.

A point pattern is defined as a series of events in a given area, or window, of observation. It is therefore extremely important to precisely define this window. In **spatstat** the function **owin** is used to set the observation window. However, the standard function takes the coordinates of a rectangle or of a polygon from a matrix, and therefore it may be a bit tricky to use. Luckily the package **maptools** provides a way to transform a *SpatialPolygons* into an object of class *owin*, using the function **as.owin** (Note: a function with the same name is also available in **spatstat** but it does not work with *SpatialPolygons*, so be sure to load **maptools**):

```
window <- as.owin(GreaterLondon)
```

Now we can use the function **ppp**, in **spatstat**, to create the point pattern object:

```
Drugs.ppp <- ppp(x=Drugs@coords[,1],y=Drugs@coords[,2],window=window)
```

We are now able to compute the G Function and generate the plot of this function for our data using the following lines:

```
jpeg("GFunction.jpeg",2500,2000,res=300)
plot(Gest(Drugs.ppp),main="Drug Related Crimes")
```

Are you able to interpret the plot?

3: Cluster Analysis: Theoretical Background

With the G function we are able to determine quantitatively that our dataset is clustered, which means that the events are not driven by chance but by some external factor. Now we need to verify that indeed there is a cluster of points located around the source of pollution, to do so we need a form of classification of the points.

Cluster analysis refers to a series of techniques that allow the subdivision of a dataset into subgroups, based on their similarities (James et al., 2013). There are various clustering methods, but probably the most common is k-means clustering. This technique aims at partitioning the data into a specific number of clusters, defined *a priori* by the user, by minimizing the within-clusters variation. The within-cluster variation measures how much each event in a cluster k, differs from the others in the same cluster k. The most common way to compute the differences is using the squared Euclidean distance (James et al., 2013), calculated as follow:

$$W_k = \frac{1}{n_k} \sum_{i=1}^{n_k} \sum_{j=1}^{p} (x_{ij} - x_{i'j})^2$$

Where W_k is the within-cluster variation for the cluster k, n_k is the total number of elements in the cluster k, p is the total number of variables we are considering for clustering and x_{ij} is one variable of one event contained in cluster k. This equation seems complex, but it actually quite easy to understand.

3.1 Practical Example

In this experiment we explore seismic events using earth quick data from the USGS (United States Geological Survey) website. Exercise preparation and code sourced from material by Veronesi (2015b).

To complete this exercise you would need the following packages:
sp, raster, plotrix, rgeos, rgdal, and scatterplot3d

First download the data using the following code:

```
URL <- "http://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_month.csv"
Earthquake_30Days <- read.table(URL, sep = ",", header = T)
```

Then Download, unzip and load the polygon shapefile with the countries' borders:

```
download.file("http://thematicmapping.org/downloads/TM_WORLD_BORDERS_SIMPL-
0.3.zip",destfile="TM_WORLD_BORDERS_SIMPL-0.3.zip")
unzip("TM_WORLD_BORDERS_SIMPL-0.3.zip",exdir=getwd())
polygons <- shapefile("TM_WORLD_BORDERS_SIMPL-0.3.shp")
```

Exploring the origin of earth quick using the cluster analysis:
We can divide the seismic events by origin to distinguish between events close to plates, or volcanoes or other faults. In many cases the distinction is hard to make since many volcanoes are originated from subduction, e.g. the Andes, where plates and volcanoes are close to one another and the algorithm may find difficult to distinguish the origins.

In this experiment we explore the use of cluster analysis to see what the algorithm is able to do.

We first need to download data regarding the location of plates, faults and volcanoes from the following website:

http://legacy.jefferson.kctcs.edu/techcenter/gis%20data/

The data are provided in zip files, so we need to extract them and load them in R. Details of the license and other information may be found here:

http://legacy.jefferson.kctcs.edu/techcenter/gis%20data/World/Earthquakes/plat_lin.htm#getacopy

If you have the rights to download and use these data for your studies you can download them directly from the web with the following code.

Create a new folder using string with the name of the folder then downloads data from the address above, unzip them and load them in R.

```
dir.create(paste(getwd(),"/GeologicalData",sep=""))

 #Faults
download.file("http://legacy.jefferson.kctcs.edu/techcenter/gis%20data/World/Zip/FAULTS.zip",destfile="GeologicalData/FAULTS.zip")
unzip("GeologicalData/FAULTS.zip",exdir="GeologicalData")
 faults <- shapefile("GeologicalData/FAULTS.SHP")

 #Plates
download.file("http://legacy.jefferson.kctcs.edu/techcenter/gis%20data/World/Zip/PLAT_LIN.zip",destfile="GeologicalData/plates.zip")
unzip("GeologicalData/plates.zip",exdir="GeologicalData")
 plates <- shapefile("GeologicalData/PLAT_LIN.SHP")

 #Volcano
download.file("http://legacy.jefferson.kctcs.edu/techcenter/gis%20data/World/Zip/VOLCANO.zip",destfile="GeologicalData/VOLCANO.zip")
unzip("GeologicalData/VOLCANO.zip",exdir="GeologicalData")
 volcano <- shapefile("GeologicalData/VOLCANO.SHP")
```

We have not yet transformed the object *Earthquake_30Days*, which is now a*data.frame*, into a *SpatioPointsDataFrame*. The data from USGS contain seismic events that are not only earthquakes but also related to mining and other events. For this analysis we want to keep only the events that are classified as earthquakes, which we can do with the following code:

```
Earthquakes <- Earthquake_30Days[paste(Earthquake_30Days$type)=="earthquake",]
coordinates(Earthquakes)=~longitude+latitude
```

This extracts only earthquakes and transforms the object into a *SpatialObject*.

We can create a map that shows the earthquakes alongside all the other geological elements we downloaded using the following code, which saves directly the image in jpeg:

```
# jpeg("Earthquake_Origin.jpg",4000,2000,res=300)
plot(plates,col="red")
plot(polygons,add=T)
title("Earthquakes in the last 30 days",cex.main=3)
lines(faults,col="dark grey")
points(Earthquakes,col="blue",cex=0.5,pch="+")
points(volcano,pch="*",cex=0.7,col="dark red")
legend.pos <- list(x=20.97727,y=-57.86364)
 legend(legend.pos,legend=c("Plates","Faults","Volcanoes","Earthquakes"),pch=c("-","-
","*","+"),col=c("red","dark grey","dark red","blue"), bty="n",bg=c("white"),y.intersp=0.75,title="Days
from Today",cex=0.8)
 text(legend.pos$x,legend.pos$y+2,"Legend:")
#dev.off()
```

Before proceeding with the cluster analysis we first need to fix the projections of the *SpatialObjects*. Luckily the object polygons was created from a shapefile with the projection data attached to it, so we can use it to tell R that the other objects have the same projection:

```
projection(faults)=projection(polygons)
projection(volcano)=projection(polygons)
projection(Earthquakes)=projection(polygons)
projection(plates)=projection(polygons)
```

Now run clustering in order classify earthquakes based on their distance between the various geological features. To calculate this distance we can use the function **gDistance** in the package **rgeos**.

These shapefiles are all unprojected, and their coordinates are in degrees. We cannot use them directly with the function **gDistance** because it deals only with projected data, so we need to transform them using the function **spTransform** (in the package **rgdal**). This function takes two arguments, the first is the *SpatialObject*, which needs to have projection information, and the second is the data regarding the projection to transform the object into. The code for doing that is the following:

```
volcanoUTM <- spTransform(volcano,CRS("+init=epsg:3395"))
faultsUTM <- spTransform(faults,CRS("+init=epsg:3395"))
EarthquakesUTM <- spTransform(Earthquakes,CRS("+init=epsg:3395"))
platesUTM <- spTransform(plates,CRS("+init=epsg:3395"))
```

The projection we are going to use is the standard Mercator (See more details at: http://spatialreference.org/ref/epsg/wgs-84-world-mercator/)

We are now creating a matrix of distances between each earthquake and the geological features We first create an empty matrix then fill the matrix using a loop. In the loop we iterate through the earthquakes and for each we calculate its distance to the geological features. Finally we change the *matrix* into a*data.frame*.

```
distance.matrix <-
matrix(0,nrow(Earthquakes),7,dimnames=list(c(),c("Lat","Lon","Mag","Depth","DistV","DistF","DistP")))

for(i in 1:nrow(EarthquakesUTM)){
sub <- EarthquakesUTM[i,]
dist.v <- gDistance(sub,volcanoUTM)
dist.f <- gDistance(sub,faultsUTM)
dist.p <- gDistance(sub,platesUTM)
distance.matrix[i,] <- matrix(c(sub@coords,sub$mag,sub$depth,dist.v,dist.f,dist.p),ncol=7)
}

 distDF <- as.data.frame(distance.matrix)
```

The next step is finding the correct number of clusters (See more details at: http://www.mattpeeples.net/kmeans.html)

The code to do that is the following:

```
mydata <-  scale(distDF[,5:7])
wss <- (nrow(mydata)-1)*sum(apply(mydata,2,var))
  for (i in 2:15) wss[i] <- sum(kmeans(mydata,
                               centers=i)$withinss)
plot(1:15, wss, type="b", xlab="Number of Clusters",
    ylab="Within groups sum of squares")
```

You are now ready to discuss the number of required clusters.

To create the clusters we can simply use the function **kmeans**, which takes two arguments: the data and the number of clusters:

```
clust <- kmeans(mydata,5)
distDF$Clusters <- clust$cluster
```

We can check the physical meaning of the clusters by plotting them against the distance from the geological features using the function **scatterplot3d**, in the package **scatterplot3d**:

```
scatterplot3d(distDF$DistV,xlab="Distance to Volcano",distDF$DistF,ylab="Distance to
Fault",distDF$DistP,zlab="Distance to Plate", color =
clust$cluster,pch=16,angle=120,scale=0.5,grid=T,box=F)
```

This function is very similar to the standard **plot** function, but it takes three arguments instead of just two. I wrote the line of code distinguishing between the three axes to better understand it. So we have the variable for x, and the corresponding axis label, and so on for each axis. Then we set the colours based on clusters, and the symbol with *pch*, as we would do in **plot**. The last options are only available here: we have the *angle* between x and y axis, the *scale* of the z axis compared to the other two, then we plot a *grid* on the xy plane and we do not plot a *box* all around the plot.

Can you make your conclusion of data using the new plot?

We can create an image with the clusters using the following code:

```
clustSP <-
SpatialPointsDataFrame(coords=Earthquakes@coords,data=data.frame(Clusters=clust$cluster))

jpeg("Earthquake_Clusters.jpg",4000,2000,res=300)
plot(plates,col="red")
plot(polygons,add=T)
title("Earthquakes in the last 30 days",cex.main=3)
lines(faults,col="dark grey")
points(volcano,pch="x",cex=0.5,col="yellow")
legend.pos <- list(x=20.97727,y=-57.86364)

points(clustSP,col=clustSP$Clusters,cex=0.5,pch="+")
legend(legend.pos,legend=c("Plates","Faults","Volcanoes","Earthquakes"),pch=c("-","-
","x","+"),col=c("red","dark grey","dark red","blue"),bty="n",bg=c("white"),y.intersp=0.75,title="Days
from Today",cex=0.6)

text(legend.pos$x,legend.pos$y+2,"Legend:")
```

We created the object *clustSP* based on the coordinates in WGS84 so that we can plot everything as before. We plotted the volcanoes in yellow, so that differ from the red cluster. The result is the following image:

You can now explore the relation between the distance to the geological features and the magnitude of the earthquakes. To do that we need to identify the events that are at a certain distance from each geological feature. We can use the function **gBuffer**, again available from the package **rgeos**, for this job.

```
volcano.buffer <- gBuffer(volcanoUTM,width=1000)
volcano.over <- over(EarthquakesUTM,volcano.buffer)
plates.buffer <- gBuffer(platesUTM,width=1000)
plates.over <- over(EarthquakesUTM,plates.buffer)
 faults.buffer <- gBuffer(faultsUTM,width=1000)
faults.over <- over(EarthquakesUTM,faults.buffer)
```

This function takes minimum two arguments, the *SpatialObject* and the maximum distance (in metres because it requires data to be projected) to reach with the buffer, option *width*. The results is a *SpatialPolygons* object that include a buffer around the starting features; for example if we start with a point we end up with a circle of radius equal to *width*. In the code above we first created these buffer areas and then we overlaid *EarthquakesUTM* with these areas to find the events located within their borders. The overlay function returns two values: NA if the object is outside the buffer area and 1 if it is inside. We can use this information to subset *EarthquakesUTM* later on.

Now we can include the overlays in EarthquakesUTM as follows:

```
EarthquakesUTM$volcano <- as.numeric(volcano.over)
EarthquakesUTM$plates  <- as.numeric(plates.over)
EarthquakesUTM$faults  <- as.numeric(faults.over)
```

To determine if there is a relation between the distance from each feature and the magnitude of the earthquakes we can simply plot the magnitude's distribution for the various events included in the buffer areas we created before with the following code:

```
plot(density(EarthquakesUTM[paste(EarthquakesUTM$volcano)=="1",]$mag),ylim=c(0,2),xlim=c(0,10)
,main="Earthquakes by Origin",xlab="Magnitude")
lines(density(EarthquakesUTM[paste(EarthquakesUTM$faults)=="1",]$mag),col="red")
lines(density(EarthquakesUTM[paste(EarthquakesUTM$plates)=="1",]$mag),col="blue")
legend(3,0.6,title="Mean magnitude per
origin",legend=c(paste("Volcanic",round(mean(EarthquakesUTM[paste(EarthquakesUTM$volcano)=="
1",]$mag),2)),paste("Faults",round(mean(EarthquakesUTM[paste(EarthquakesUTM$faults)=="1",]$ma
g),2)),paste("Plates",round(mean(EarthquakesUTM[paste(EarthquakesUTM$plates)=="1",]$mag),2))),
pch="-",col=c("black","red","blue"),cex=0.8)
```

Are you now able to see any pattern within data?
It seems that earthquakes close to plates have higher magnitude on average what do you think?

This exercise has been compiled from various important resources, with each duly noted in the text.

References:

Baddley, A. (2010) Analysing spatial point patterns in R. Workshop Notes Version 4.1, CSIRO. Online, viewed 31 August 2016, http://research.csiro.au/software/wp-content/uploads/sites/6/2015/02/Rspatialcourse_CMIS_PDF-Standard.pdf

Han, J., Kamber, M. and Tung, K. H. (2001) Spatial Clustering Methods in Data Mining: A Survey. In Harvey J. Miller and Jiawei Han (eds.), *Geographic Data Mining and Knowledge Discovery*, CRC Press. Boca Raton, FL, USA.

Ervin, D. (2016) Point pattern analysis. Advanced Spatial Analysis, University of California Santa Barbara. Online, viewed 31 August 2016, http://gispopsci.org/point-pattern-analysis/

Veronesi, F. (2015a) Introductory point pattern analysis of open crime data in London. R Tutorial for Spatial Statistics. Online, viewed 31 August 2016, http://r-video-tutorial.blogspot.com.au/2015/05/introductory-point-pattern-analysis-of.html

Veronesi, F. (2015b) Cluster analysis on earthquake data from USGS. R-bloggers. Online, viewed 31 August 2016, http://www.r-bloggers.com/cluster-analysis-on-earthquake-data-from-usgs/