R package **`plotGoogleMaps`** for automatic creation of web maps – map mashups over Google Maps

Milan Kilibarda[1]

[1] University of Belgrade, Faculty of Civil Engineering, Department of Geodesy and Geoinformatics, Bulevar kralja Aleksandra 73 11000 Belgrade, Serbia, kili@grf.bg.ac.rs

Contents:

1 Introduction

The `plotGoogleMaps` package provides an interactive plot device for handling the geographic data within web browsers. It is optimized for Google Chrome browser. It is designed for the automatic creation of web maps as a combination of users' data and Google Maps layers.
The input data are in form of Spatial-class with associated coordinate reference system. The classes and methods for spatial data and its manipulation is described in book *Applied Spatial Data Analysis with R* (Bivand at al, 2008).
The `plotGoogleMaps` is based on Google Maps API. Google Maps API is set of predefined JavaScript classes ready to be implemented in any web page, with aim of creation interactive web map – Google map mashup. It is possible to create map mashup even if creator is not an expert in web programming, although the basic knowledge in JavaScript programming language, XML, Ajax and XHTML is required.
The `plotGoogleMaps` enables creation of interactive web map, with the base map supplied by Google, where all map elements and additional functionalities are handled by just one R command from the package. The package provides solution to create and visualize vector and raster data, proportional symbols, pie charts and ellipses. The web map – map mashup created by `plotGoogleMaps` package could be used as a temporary result of spatial visualization generated on the local machine or, published on any web page.
The `plotGoogleMaps` uses web browser as plotting device instead of default R graphic device. Therefore, it offers more advantages related to R classical plotting device environment; high quality of background Google layers which make better abstraction of geographical reality, spatial data exploration functionality, and map interactivity (navigation control, pan, zoom, attribute info windows, etc). Google Maps API is not suitable for the large data and consequently `plotGoogleMaps` has the same constrain (Kilibarda and Bajat, 2012).
This vignette describes functions provided by `plotGoogleMaps`.
Package `plotGoogleMaps` is loaded by:

```
library(plotGoogleMaps)
```

2 Plotting spatial points with `plotGoogleMaps`

In the following example, it is shown plot of SpatialPointsDataFrame objects of meuse data set. This data set gives locations and topsoil heavy metal concentrations of 155 observations together with soil properties and distance to the river, collected in a flood plain of the river Meuse, in the area around Meers and Maasband (Limburg, the Netherlands) (N=50°58′16′′ E=05°44′39′′) during a fieldwork in the year 1990.

```
# Data preparation
# Point data
data(meuse)
coordinates(meuse)<-~x+y # convert to SPDF
proj4string(meuse) <- CRS('+init=epsg:28992')
# adding Coordinate Referent Sys.

#Create web map of Point data
m<-plotGoogleMaps(meuse,filename='myMap1.htm')
```

The first created map is named myMap1.htm , and it is map of meuse data, mashup with positions of points and attribute data ready to be explored in browser.

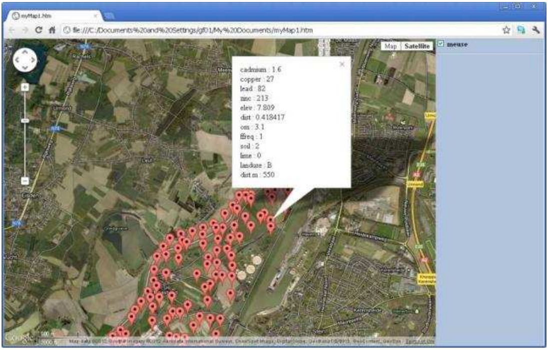

Figure 1: plot of SpatialPointsDataFrame object; meuse data

In the next example some additional setting for the `plotGoogleMaps` is presented.

```
m<-plotGoogleMaps(meuse,
filename='myMap2.htm',
iconMarker='http://maps.google.com/mapfiles/kml/shapes/placemark_circle.png',
mapTypeId='ROADMAP',
layerName = 'MEUSE POINTS')
```

By using `iconMarker` attribute, it is set custom marker image from local disk or from Web. In this case, it is marker from Google Earth KML gallery images. It is easy to change Google Maps layer which is active after exploring htm file by controlling mapTypeId and argument layer name in htm by using `layerName`. See Figure 2.

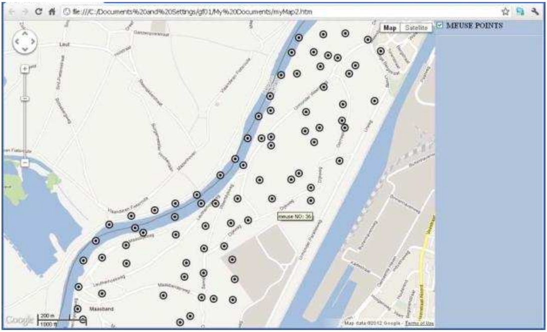

Figure 2: plot of SpatialPointsDataFrame object; meuse data; with additional settings

The sampled zinc concentration can be plot with proportional symbols and in different colors related to measured concentration. Maximum radius related to maximum concentration is specified in meters.

```
m<-bubbleGoogleMaps(meuse,zcol='zinc', max.radius = 80,
filename='myMap3.htm')
```

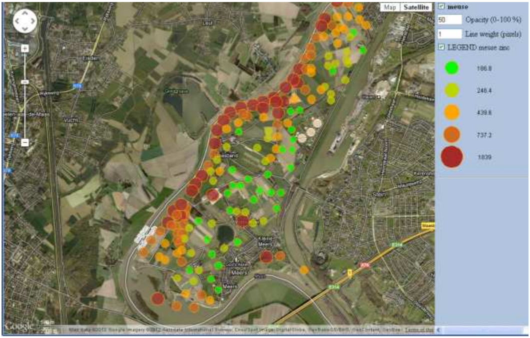

Figure 3: plot of SpatialPointsDataFrame object; meuse data; bubbleGoogleMaps

The function segmentGoogleMaps produces maps for multivariate mapping. The segmentGoogleMaps creates pie charts or more properly called segmented circles. Pie charts are circles with wedges representing the variables, which are related in some way. In ths example it is presented multivariate plot of heavy metal concentrations from meuse sampling points. Maybe it should be more properly to present some variables that are more related.

```
data(meuse)
coordinates(meuse)<-~x+y
proj4string(meuse) <- CRS('+init=epsg:28992')

m<-segmentGoogleMaps(meuse, zcol=c('zinc','lead','copper'),
mapTypeId='ROADMAP', filename='myMap4.htm')
```

In the `zcol` argument is set variables to be presented in pie chart manner. If data contains just that variable it isn't necessary to be set.

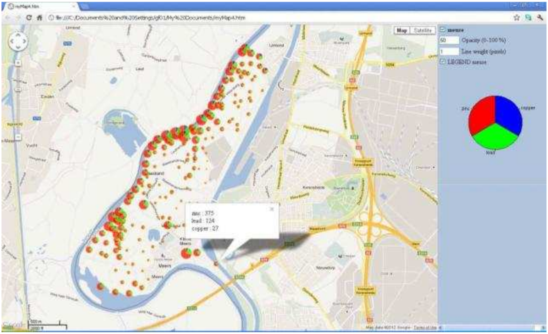

Figure 4: plot of SpatialPointsDataFrame in form of pie charts plot.

Plotting uncertainty of position is provided by `ellipseGoogleMaps` function. The `ellipseGoogleMaps` plots standard errors of the computed coordinates, error ellipses describing the uncertainty of a two-dimensional position. Parameters of input spatial points data frame should contain at least three columns: semi-major axis, semi-minor axis, and orientation in degrees. These parameters are product of geodetic least square adjustment or design of a geodetic control network.
In the next example is shown results from geodetic network design results.

```
# Results of least square
```

```
ell<- data.frame(E=c(7456263,7456489,7456305,7457415,7457688),
N=c(4954146 ,4952978, 4952695, 4953038, 4952943),
Name=c('30T', '31T', '3N', '40T', '41T'),
A=c(2.960863 ,4.559694, 7.100088, 2.041084 ,3.375919),
B=c(2.351917, 2.109060, 2.293085, 1.072506, 2.382449),
teta=c(28.35242, 41.04491, 38.47216, 344.73686, 27.53695))

coordinates(ell) <- ~E+N

proj4string(ell) <- CRS("+proj=tmerc +lat_0=0 +lon_0=21 +k=0.9999
+x_0=7500000 +y_0=0 +ellps=bessel
+towgs84=574.027,170.175,401.545,4.88786,-0.66524,-
13.24673,0.99999311067 +units=m")

m<-ellipseGoogleMaps( ell, filename="Ellipse.htm", zcol=2:4,
mapTypeId='ROADMAP')
```

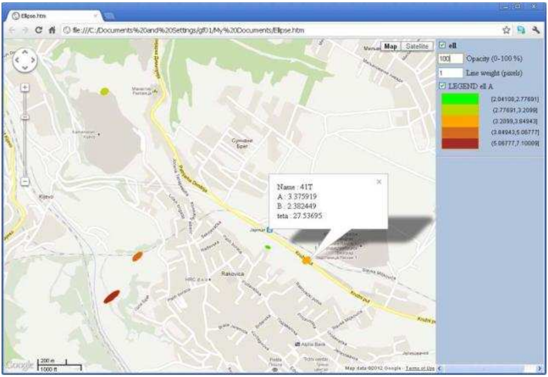

Figure 5: plot of error ellipses.

3 Plotting spatial lines with `plotGoogleMaps`

The `plotGoogleMaps` produces plot of SpatialLinesDataFrame similary like plotting SpatialPointsDataFrames. In the next example coloring is used by default and border width is set related to line attribute. The lines used in this case representing distance to Meuse River.

```
# Line data
data(meuse.grid)
coordinates(meuse.grid)<-c('x','y')
meuse.grid<-as(meuse.grid,'SpatialPixelsDataFrame')
im<-as.image.SpatialGridDataFrame(meuse.grid['dist'])
cl<-ContourLines2SLDF(contourLines(im))
proj4string(cl) <- CRS('+init=epsg:28992')
mapMeuseCl<- plotGoogleMaps(cl, zcol='level', strokeWeight=1:9 ,
filename='myMap5.htm', mapTypeId='ROADMAP')
```

The strokeColor argument defines line width corresponding to line attribute `level`, distance to river.

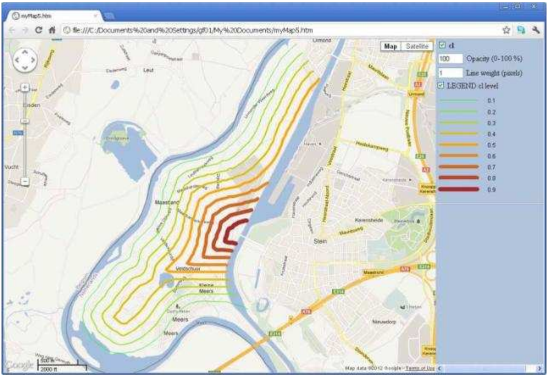

Figure 6: plot of SpatialLinesDataFrame

4 Plotting spatial polygon data with `plotGoogleMaps`

For plotting spatial polygon data frame is used shapefile provided by `maptools` package. This is for the 100 counties of North Carolina, and includes counts of numbers of live births (also non-white live births) and numbers of sudden infant deaths, for the July 1, 1974 to June 30, 1978 and July 1, 1979 to June 30, 1984 periods (Bivand, 2011).
For the colour coding `RColorBrewer` package is used.

Next command plots nc data with colour sheme obtained from `RColorBrewer` for the polygons and white border is set to county border. The color scheme relates to plotting attribute named `BIR74`.

```
nc <- readShapeSpatial( system.file("shapes/sids.shp",
package="maptools")[1], proj4string=CRS("+proj=longlat
+datum=NAD27"))

library(RColorBrewer)

plotGoogleMaps(nc,
zcol="NWBIR74",
filename='MyMap6.htm',
mapTypeId='TERRAIN',
colPalette= brewer.pal(7,"Reds"),
strokeColor="white")
```

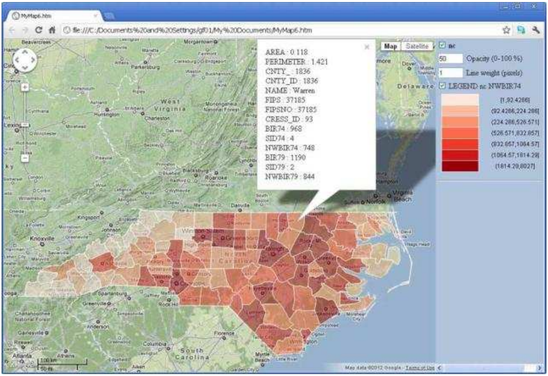

Figure 7: plot of SpatialPolygonsDataFrame

5 Plotting spatial grid/pixels data with `plotGoogleMaps`

In the next example is shown plot SpatialPixelsDataFrame.

```
data(meuse.grid)
coordinates(meuse.grid)<-c('x','y')
```

```
meuse.grid<-as(meuse.grid,'SpatialPixelsDataFrame')
proj4string(meuse.grid) <- CRS('+init=epsg:28992')

plotGoogleMaps(meuse.grid, zcol='dist', mapTypeId='ROADMAP')
```

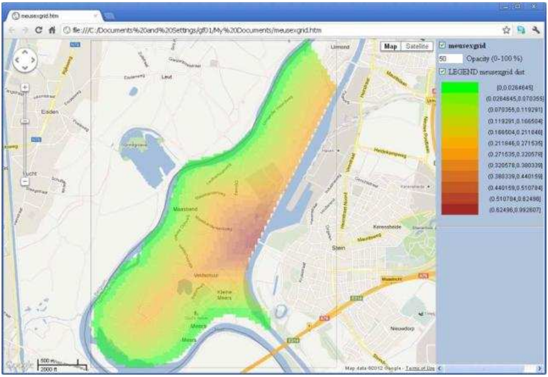

Figure 8: plot of SpatialPixelsDataFrame

6 Combining several layers with `plotGoogleMaps`

A map becomes more readable when is combined several layers. The `plotGoogleMaps` functions could be use to create map mashup with several layers, the function should contain argument `add=TRUE.` The next plot should have the name of previous map the argument `previousMap = <name of saved map produced by functions from plotGoogleMaps package> .`

```
m1<- plotGoogleMaps(cl, zcol='level', strokeWeight=1:9 , add=
TRUE)

m2<-bubbleGoogleMaps(meuse,zcol='zinc', add=T,
colPalette= brewer.pal(5,"Accent"),
max.radius = 80,previousMap= m1)
```

```
m3<- plotGoogleMaps(meuse.grid, zcol='dist',colPalette=
brewer.pal(6,"Oranges"),previousMap= m2,
filename='combination.htm')
```

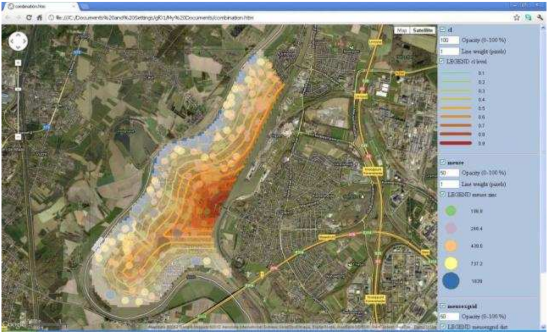

Figure 9: plot several layers

References:

Bivand, R. S., Pebesma, E. J. and Gomez-Rubio, V. 2008. *Applied Spatial Data Analysis with R*. Springer , New York, 378 p.

Bivand, R. S. 2011. Introduction to the North Carolina SIDS data set (revised) , http://cran.r-project.org/web/packages/spdep/vignettes/sids.pdf

Kilibarda, M. and Bajat, B. 2012. plotGoogleMaps: The R-based web-mapping tool for thematic spatial data. GEOMATICA Vol. 66, No. 1, (2012), p. 37-49