Journal of Statistical Software

January 2015, Volume 63, Issue 4. http://www.jstatsoft.org/

RgoogleMaps and loa: Unleashing R Graphics Power on Map Tiles

Markus Loecher
Berlin School of Economics and Law

Karl Ropkins
University of Leeds

Abstract

The **RgoogleMaps** package provides (1) an R interface to query the Google and the OpenStreetMap servers for static maps in the form of PNGs, and (2) enables the user to overlay plots on those maps within R. The **loa** package provides dedicated panel functions to integrate **RgoogleMaps** within the **lattice** plotting environment.

In addition to solving the generic task of plotting on a map background in R, we introduce several specific algorithms to detect and visualize spatio-temporal clusters. This task can often be reduced to detecting over-densities in space relative to a background density. The relative density estimation is framed as a binary classification problem. An integrated hotspot visualizer is presented which allows the efficient identification and visualization of clusters in one environment. Competing clustering methods such as the scan statistic and the density scan offer higher detection power at a much larger computational cost. Such clustering method can then be extended using the **lattice** trellis framework to provide further insight into the relationship between clusters and potentially influential parameters. While there are other options for such map 'mashups' we believe that the integration of **RgoogleMaps** and **lattice** using **loa** can in certain circumstances be advantageous, e.g., by providing a highly intuitive working environment for multivariate analysis and flexible testbed for the rapid development of novel data visualizations.

Keywords: scan statistic, **RgoogleMaps**, **loa**, hotspots, supervised learning, PRIM, lattice, conditional clusters.

1. Introduction

This paper summarizes recent advances in the conditional visualization of spatial/spatio-temporal data as implemented by the R packages **RgoogleMaps** (Loecher 2014) and **loa** (Ropkins 2014). We further suggest that the computationally demanding task of searching for regions of very high or very low count density can be expedited by supervised learning tech-

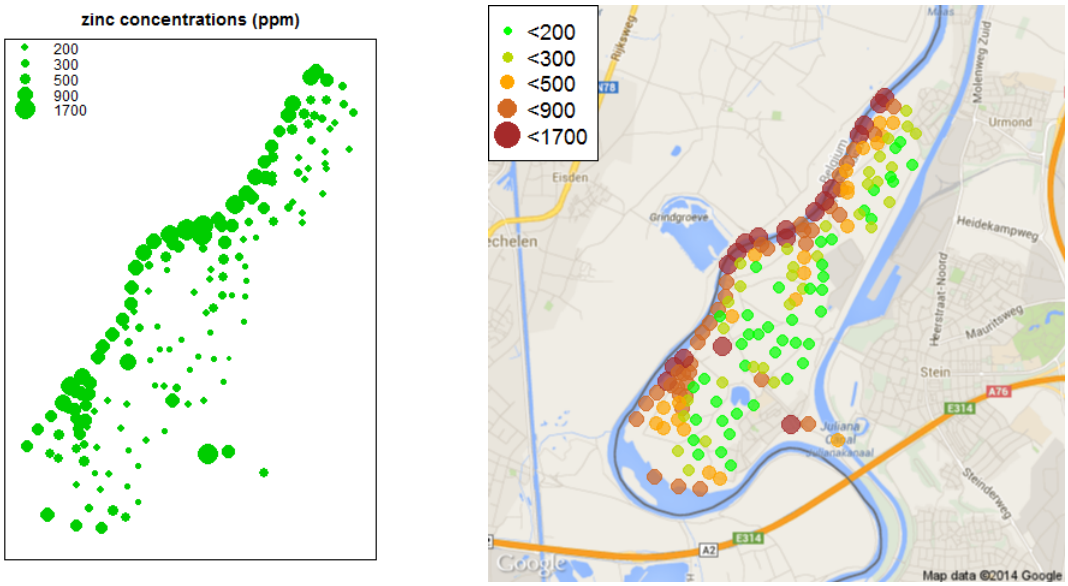

Figure 1: Left: meuse data set visualized by the `bubble()` command in the **sp** package (Pebesma and Bivand 2005; Bivand *et al.* 2013). Right: meuse data set visualized by the commands `data("lat.lon.meuse", package = "loa")` and `bubbleMap(lat.lon.meuse, coords = c("longitude", "latitude"), zcol = "zinc")` in the R package **RgoogleMaps**.

niques such as classification trees and generalized additive models. The motivation to want to draw on map tiles directly within R (R Core Team 2014) is twofold. While the R environment boasts a long history of spatial analysis tools and packages, the plots have traditionally been created without any spatial context that a map would provide. Figure 1 shows as an example two different ways of plotting the *meuse* (Rikken and Van Rijn 1993) data set. (As with all later figures, R scripts for Figure 1 are provided in the supporting material for this paper.) There are times when the left plot will be the preferred choice: a clean and simple graph of the locations of interest with no clutter and no distractions. The analyst can focus on the pattern itself and the marker attributes.

However, a lot of the modern massive data sets gathered such as location information from mobile devices, surveys, crime data, vehicle tracks, demographic data, etc. require a map based spatial context for even the most basic data explorations. In those settings, the somewhat narrow or "blind" exploration of spatial data on a blank background can be rather limiting and often leads to less insight than would be possible had the data been graphed on a map canvas. Anecdotally, we dare to speculate that the British physician John Snow might have been slower to identify contaminated drinking water as the cause of the cholera epidemic in 1854, had he not used a dot map to illustrate the cluster of cholera cases around one of the pumps in London (Johnson 2006).

While there exist many HTML and/or GIS based solutions to this simple problem, the overhead of switching tools and environments can be detrimental to efficient development. In addition, the user often has to obey a number of constraints with respect to number of data points, size and shape of the markers or polygons, etc. The Google Static Maps API (Google Developers 2014b), for example, allows easy download of Google Maps images without requiring JavaScript. The Google Static Map service creates a map based on URL parameters

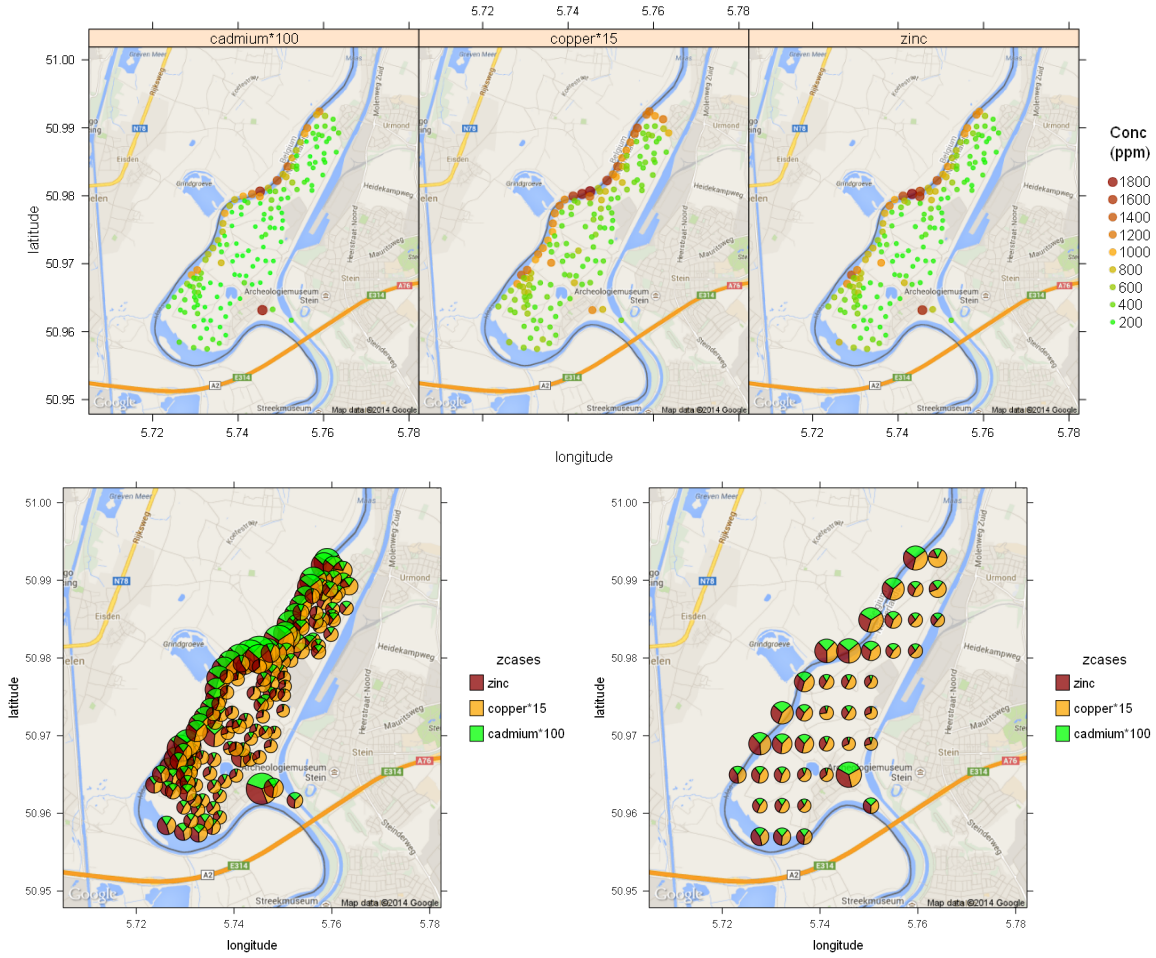

Figure 2: Top: The by-panel visualization of multiple heavy metals in the meuse data set using the `GoogleMap()` command in **loa**. Bottom left: Single panel visualizations of such information can easily become cluttered or 'over-plot', even with moderately sized data sets, as in the case of the first pie plot. Bottom right: Merging data binning and pie plot functions, as illustrated in supporting code for this paper, provides a simple 'work around'.

sent through a standard HTTP request and returns the map as an image in a wide range of formats and color schemes. For a complete list of parameters such as zoom level, center, size, map type and style and other options we refer the reader to the online documentation (Google Developers 2014b). But URL calls are restricted to 2048 characters in length which in practice can be very limiting when adding additional data.

The **RgoogleMaps** work flow for creating spatial overlays in R is broken down into two steps: Firstly, the function `GetMap()` – or its close cousin `GetMap.bbox()` – fetches the appropriate map png image from the chosen server (Google or OpenStreet Maps) and stores it along with meta information in a list structure. Secondly, **RgoogleMaps** plot functions, such as `bubbleMap()` as illustrated in Figure 1, `PlotOnStaticMap()` as illustrated in Figure 4 and `ColorMap()` as illustrated in Figure 5, merge this map and supplied data layers to generate an appropriate georeferenced visualization. The incorporation of the trellis plotting frame-

work described by Cleveland and colleagues (Cleveland 1993; Becker, Cleveland, and Shyu 1996) and implemented in **lattice** (Sarkar 2008) provides an elegant and flexible means of introducing multivariate information. The use of **loa** to integrate **RgoogleMaps** and **lattice** is illustrated in Figure 2, which uses trellis paneling to introduce additional data series to the previous bubble plot (Figure 1) first as discrete maps, and then with combined panel functions to overcome over-plotting issues 'on the fly' when merging such information on a single map. Whilst the recent package **ggmap** (Kahle and Wickham 2013a) elegantly embeds most of **RgoogleMaps**' abilities into the **ggplot2** (Wickham 2009) framework and provides some obvious advantages, e.g., a larger range of map types and geo-coding (Kahle and Wickham 2013b), we believe the inherent flexibility in the **lattice** framework makes it a particularly attractive working environment for those looking to develop novel visualization methods or rapidly modify existing plots to their own (sometimes unique) requirements.

Note that **RgoogleMaps** focuses exclusively on static map display within R and does *not* offer any dynamic map mashup capabilities. For such facilities we refer the interested reader to the packages **plotGoogleMaps** (Kilibarda and Bajat 2012) and **googleVis** (Gesmann and de Castillo 2011) instead.

1.1. Google Maps limitations

We give a brief overview of the practical and legal limitations of using Google map tiles in reports, Web sites or other applications. For more detail, we refer the reader to (Google Developers 2014a,b). Besides the above mentioned URL length, the Google Static Maps API has the following usage limits:

- When using an API key: 25000 Static Maps requests per 24 hour period.
- Without an API key: 1000 Static Maps requests per IP address per 24 hour period. 50 Static Maps requests per IP address per minute. This means that if you have a single page containing more than 50 maps, the page will exceed this limit.

The maximum size of the map tiles in pixels is 640×640 (or 1280×1280 if the parameter scale is set to 2) for the free API. The business API delivers map tiles up to 2048×2048 pixels. We refer the reader to Potere (2008) for details on the horizontal accuracy of the maps.

Legal restrictions include: (i) No unauthorized copying, modification, creation of derivative works, or display of the content. (ii) No pre-fetching, caching, or storage of content. (iii) No mass downloads or bulk feeds of content.

1.2. San Francisco crime data

The following examples in this paper use a data set of reported police incidents (excluding homicide and manslaughter) from San Francisco in 2012. The full data set is freely available for download (City and County of San Francisco 2013) and contains date, time of day, and day of week for each incident, along with the incident category, a brief description, and location (as police district, lat-long coordinates, and address to the nearest 100 block). Following WinVector (2013) we define the following crimes as violent: assault, robbery, rape, kidnapping, and purse snatching, which leads to an overall proportion of violent crime of about 12%.

2. Temporal view of the data

Before presenting a spatial analysis of the crime data, it is informative to get an overview of its temporal patterns such as long term trends, seasonality as well as hour-of-day and day-of-week signatures. Figure 3 (top) displays the daily crime counts for the last ten years along with the proportion of violent crimes. We observe an almost linear downward trend in overall crime until a structural change appears around May 2012, where the trend is reversed after another drop. Many of the dominant peaks/outliers can be attributed to holidays such as New Year's Eve, July 4th, etc. The proportion of violent crimes (measured in percent), however, remains relatively stable over the same period and a decomposition of the time series revealed no strong seasonal component.

What is the pattern of crime as one cycles through the hours of the weekdays and the weekend? Crime is generally higher on the weekends, beginning with Friday and is lowest during the late hours of the night (3am–5am). The hour with the largest number of crimes reported is Sunday morning around 1am. The dependence on hour-of-day is very strong. Figure 3 (bottom) displays the results of a binomial generalized additive model (GAM). The peak of the proportion of violent crimes around 3am on the weekends coincides with the lowest volume of crimes. The interaction of hour-of-day and day-of-week reveals a marked weekend surge.

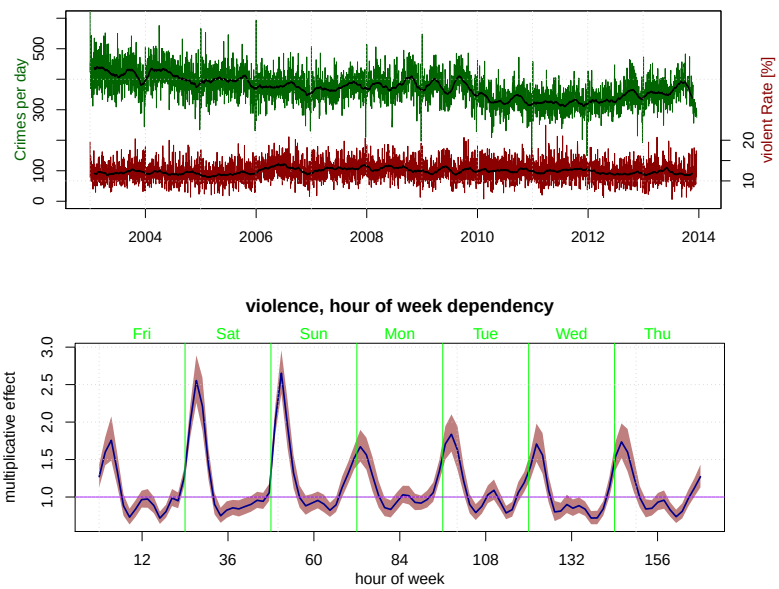

Figure 3: Top: Daily volume of crimes and rate of violence in San Francisco between 2003 and 2014. A moving average (sliding window size of 60 days) is overlaid in black. No marked seasonality is discernible. Overall crime seemed on the decline from 2003 until May 2010 where a change point appears: the volume drops rather abruptly and has been slowly rising ever since. The proportion of violent crimes (measured in percent), on the other hand shows no such trends. Bottom: Weekhour pattern of the *rate* of violent crimes. Shown is the multiplicative effect on the baseline as a function of weekhour as estimated by a generalized additive model (package **mgcv**). We notice a strong day-of-week effect.

3. Spatial view of the data

The three core functions within **RgoogleMap** are:

1. `GetMap()` fetches a map tile from the Google API. This returns a list containing the image as well as all parameters needed to properly scale coordinates in form `map <- GetMap(...)`.

2. `PlotOnStaticMap()` adds point type data to an existing map in form `PlotOnStaticMap(map, lat, lon, ...)`. Note that the full graphical power of R could be utilized in this step, i.e., hundreds of thousands of points or lines in any possible style and color and size can be overlaid.

3. `PlotPolysOnStaticMap()` adds polygon type data stored in a data structure defined by the package **PBSmapping** (Schnute, Boers, Haigh, Grandin, Johnson, Wessel, and Antonio 2014) to an existing map in form `PlotPolysOnStaticMap(map, poly, ...)`.

To illustrate this process Figures 4 and 5 were generated using the San Francisco dataset and these functions using full code provided in the supporting information for this paper.

In closing this section we point out that while **RgoogleMaps** focuses on plotting spatial information that is encoded in simple classes such as vectors, matrices and data frames, most of

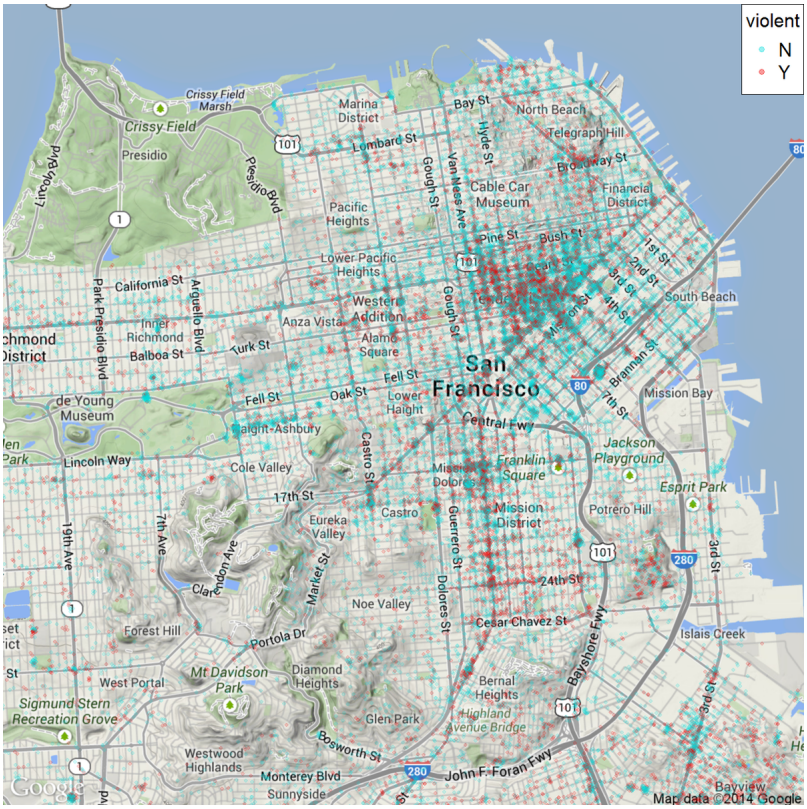

Figure 4: Crimes color coded by violent (red) and non-violent (cyan) category overlaid on a map with the **RgoogleMaps** package.

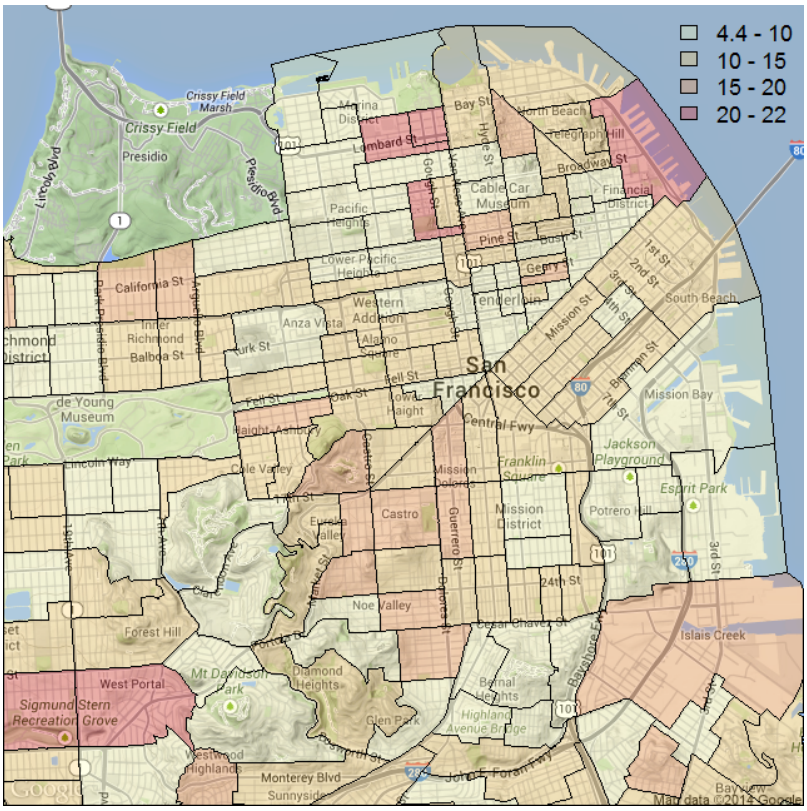

Figure 5: Rate of violent crime spatially aggregated at the census block level. The legend color codes are in percent units. Note that we applied empirical Bayes smoothing (package **epiR**) to avoid spikes due to low counts.

its functions can also handle **sp** classes directly. In particular, `SpatialPoints(DataFrame)`s, `SpatialPolygons(DataFrame)`s and `SpatialLines(DataFrame)`s can be overlayed on a map canvas. At this point we expect the user take care of the projection attribute, i.e. to first project such data back to latitude, longitude values.

3.1. Adding conditional paneling

Once we start to observe clusters or other spatial features within a data set, one of the first questions is what drives the formation of these clusters? For example, were higher densities of violent crimes associated with particular types of areas or times-of-day or times-of-week in the San Francisco data set? In Figure 2 (top) we used trellis panels to compare the concentrations of three different heavy metals measured in sediments in the Meuse area. However, we could just as easily use the trellis structure to subset a single data series and generate discrete plot panels for each data subset. This approach, known as conditional paneling, can then be used investigate such complex interactions.

The **loa** function `GoogleMap()` uses **lattice**-style plot definition by formulae `z ~ lat * lon`, where `z` values are plotted on (longitude, latitude) axes. Conditional paneling can be incorporated into a plot by adding one or more conditioning term, e.g., `z ~ lat * lon |cond1 + cond2`, etc. An example of this type of extended conditioning is provided as Figure 6 (top),

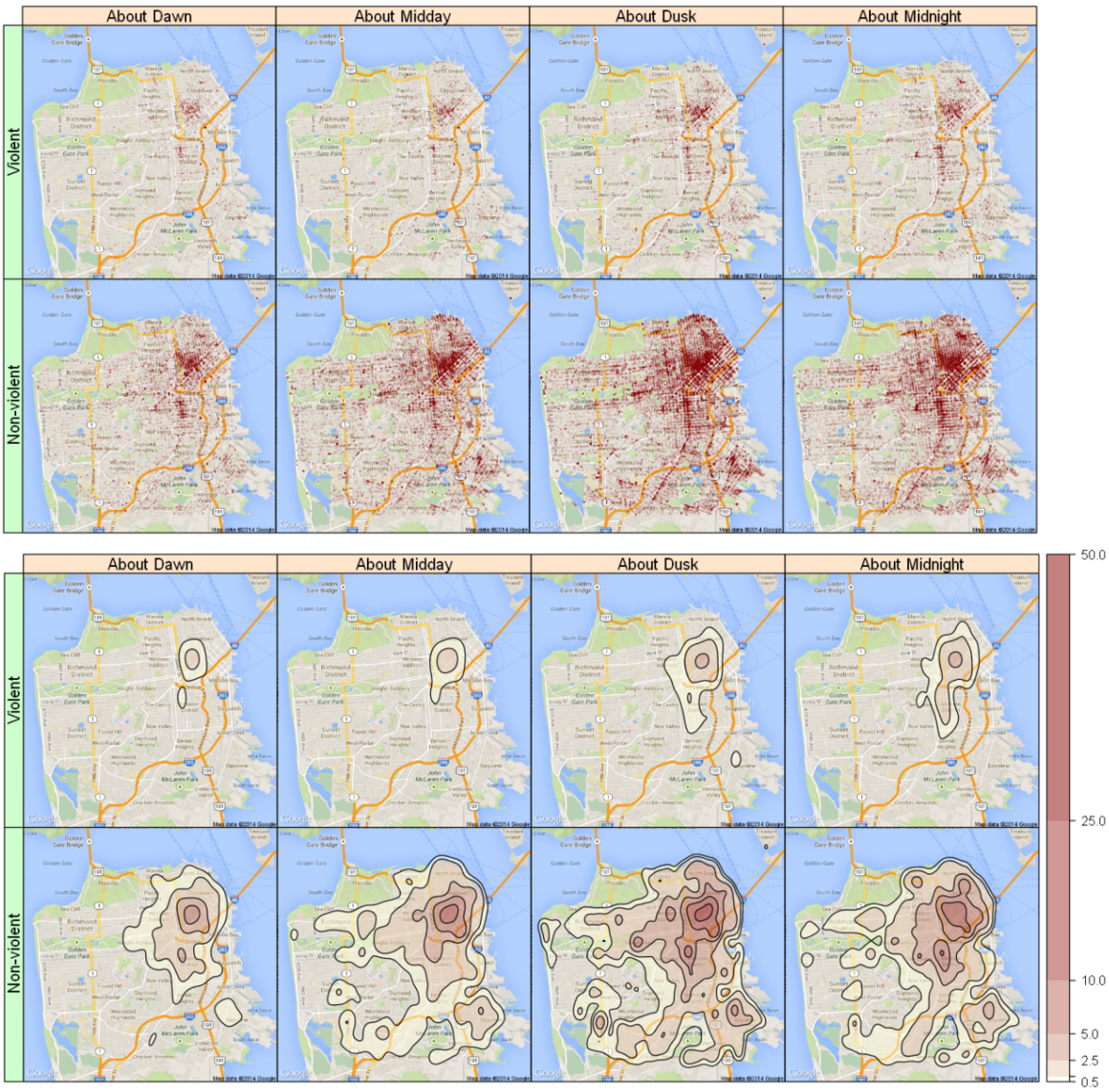

Figure 6: The 2012 San Francisco incidents dataset plotted using **loa** `GoogleMap()` and crime type (violent versus non-violent) and time-of-day as conditioning variables. Note the use of non-standard aggregation (about dawn, about midday, about dawn, about midnight). Top: A point plot using default `GoogleMap` panel settings. Bottom: A density surface plot using `panel.kernelDensity()`.

which shows the full 2012 San Francisco data set conditioned by both crime type (violent vs non-violent) and time of day (aggregated into four 6-hour intervals).

Note, while some other packages provide similar capabilities, e.g., retrospective paneling by `facet()` in **ggplot2** (Wickham 2009), the formula structure of **lattice** is perhaps one of the most intuitive and flexible.

An initial inspection of this plot suggests that there is a lot more non-violent crime than violent crime and that there is more crime at night than in the day time. However, it is

likely that several of the panels in this plot are subject to some degree of over-plotting, and this hinders further interpretation of, e.g., the similarity of clusters in the non-violent about dusk and non-violent about midnight subsets of data. Therefore, we replot the information as a density surface plot to generate a less ambiguous visualization, Figure 6 (bottom). For this, we modify the plot call used to generate the previous plot by adding the **loa** function `panel.kernelDensity()` as the `panel` argument. `panel.kernelDensity()` combines `kde2d()` in the **MASS** package (Venables and Ripley 2002) and the **lattice** function `panel.contourplot()` to generate a kernel-density based frequency surface for the number of incidents.

One of the acknowledged limitations of **lattice** is that it includes no mechanism for formal panel-to-panel and panel-to-legend communication. If values are calculated within panels, as here with the density surfaces generated by the `panel.kernelDensity()` function, ranges would not be known by other panels or any keys preventing the synchronization of associated color schemes and sizing. The hexagonal binning package **hexbin** (Carr 2014) includes the method `hexbinplot` which provides a (single output) solution for this issue by retrospectively merging the results of all the hexbin counts made in individual panels and updating the plot using this 'all counts' range before passing the plot on to the user. (See relevant discussion in Chapter 14 of Sarkar (2008) or open up `hexbinplot()`, e.g., using `edit(hexbin:::hexbinplot.formula)`, and have a look at *maxcnt* handling.) The package **loa** (Ropkins 2014) includes the function `panelPal()` that provides a generic extension of this solution to multiple output panel-to-panel and panel-to-legend communication. `GoogleMap()`, like other **loa** plot functions, uses `panelPal()` to automatically synchronize plot panels and keys.

3.2. Efficient spatial polygon search via kd trees

Given spatial partitions such as census blocks, ZIP codes or police district boundaries, we are frequently faced with the need to spatially aggregate data, e.g., in order to create views such as Figure 5. Unless efficient data structures are used, this can be a daunting task. In this case the crime data for 2012 contains more than 120K location records, each one necessitating a spatial polygon matching. The operation `point.in.polygon()` from the package **sp** (Pebesma and Bivand 2005) is computationally expensive and its use should be minimized. A brute force search would require us to execute this function for a large fraction of the appr. 200 polygons in the San Francisco area. Instead we exploit kd trees as an efficient nearest neighbor search algorithm to dramatically reduce the effective number of polygons being searched. In particular, our *massive point-in-polygon search strategy* consists of the following steps:

1. Compute the centroids for each polygon using the function `calcCentroid()` from the package **PBSmapping**.
2. Find the 10 nearest centroids for each crime location using the function `nn2()` from the package **RANN** (Kemp and Jefferis 2014).
3. In this ordered search list we then execute `point.in.polygon()` until we find a match.

On average we need to look up less than 2 polygons per point. The recent package **RapidPolygonLookup** (Loecher and Kumar 2014) on CRAN (Comprehensive R Archive Network) implements the ideas outlined above.

As variations on the proportion of violent crimes depicted in Figure 5 one could compute various baselines for the census blocks such as area or total population and display the density of crimes relative to that reference. In addition, demographic covariates available at the census block level could be used in spatial models explaining crime.

The census information (polygons and demographic attributes) are obtained from the package **UScensus2010** (Almquist 2010).

As the distribution of crimes is very heterogeneous, there are many census blocks with low counts, which makes rate estimation notoriously difficult. One solution to this dilemma is generally referred to as *shrinkage*: we estimate a prior distribution on the rate from all data and compute the posterior rate for each polygon as a weighted average of the measured proportion and the prior. We use the function `epi.empbayes()` from the package **epiR** (Stevenson 2014) for this purpose.

4. Hot spot analysis

Hot spots which are usually thought of as relatively compact areas of "high intensity" can be defined in various ways; most important here is the definition of the baseline. We might be interested in simply the density of crimes or locations of a high rate of, for example, violent crimes or vehicle theft or some other category of crime. We can also define the baseline as historical counts and look for deviations from the regular patterns that have been established. The variations on what constitutes the background are as diverse as the applications needing them.

Before we let algorithms identify and isolate spatial clusters, it is informative to manually pick two areas of visibly high crime volume and inspect them in detail. We chose the strikingly dense cluster below the bend of I-80 as well as the elongated region along Mission Street in the Mission District, both of which are displayed in Figure 7. The insight gained from the contextual information of the map tile is particularly useful for the first cluster on Bryant street which is located right adjacent to the San Francisco (SF) police department "field operations" as well as the SF county jail. In order to provide a multifaceted view of the data in a cluster, we overlay the violent crime crate rate as a function of time-of-day using the `par(fig)` method of creating insets. Such extra graphs can be used to compare with either the overall patterns or other hot spots.

4.1. Spatial cluster detection

Monitoring spatially and temporally varying activity of various kinds is an important tool for many technologies and scientific disciplines. The spatial scan statistic (Kulldorff 1997, 2001; Kulldorff, Heffernan, Hartman, Assuncao, and Farzad 2005) and its associated public domain software SatScan (Kulldorff and Information Management Services Inc. 2014) are widely used for the detection and evaluation of disease clusters.

Here, instead we follow an idea proposed by (Hastie, Tibshirani, and Friedman 2009, pp. 594–501) by transforming the density estimation problem into one of supervised function approximation.

In earlier work (Loecher 2012) we chose to apply two very different learning algorithms to a simulated Gaussian bump embedded in uniform background data. The first natural choice is

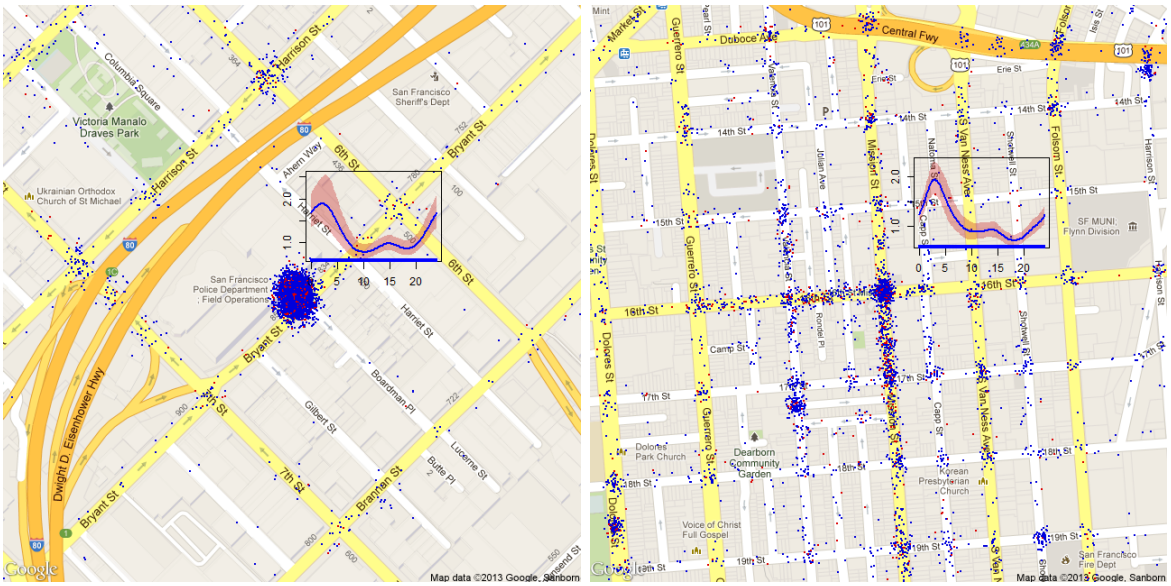

Figure 7: Left: One "hot spot" found where the contextual information provided by the map is invaluable. This cluster of crimes is found right at 850 Bryant Street, which happens to be adjacent to the SF police department "field operations" as well as the SF county jail. We have overlaid the violent crime crate rate as a function of time-of-day (compare to Figure 3 bottom). Right: Another cluster of crime activity spread along the street grid.

classification trees (Breiman, Friedman, Olshen, and Stone 1984) as implemented by the R package **tree** (Ripley 2014): A tree is grown by binary recursive partitioning using the class label and choosing splits from the two spatial coordinates. Numeric variables are divided into $X < a$ and $X > a$; the split which maximizes the reduction in impurity is chosen, the data set split and the process repeated. Splitting continues until the terminal nodes are too small or too few to be split. For illustration purposes we pretend that these artificial data were measured in the Manhattan area and plot them on a map background as shown in Figure 8 left.

We also plot the partitions found by the binary recursive algorithm and find that the location and extent of the spatial cluster is identified rather accurately. The labels indicate the fraction of the positive class labels found in the respective rectangle. Note that the computational time needed to identify this cluster constitutes a tiny fraction of the exhaustive search conducted by both **SaTScan** (Kulldorff and Information Management Services Inc. 2014) and even the "fast scan statistic" (Neill 2012). There are no guarantees that all significant clusters are found but in many situations that would be considered a fair tradeoff.

Tree-based methods try to make the response averages in each box as different as possible. The most common choices for the cost functions used when growing and pruning the tree are the misclassification error, the Gini index and the cross entropy (Breiman *et al.* 1984; Hastie *et al.* 2009). We believe that the tree growing/pruning algorithm could be easily extended to optimize a score function instead.

The *patient rule induction method* (PRIM) as outlined in (Hastie *et al.* 2009, pp. 317–320) also finds boxes in the feature space, but seeks boxes in which the response average is high. The main box construction method in PRIM works from the top down, starting with a box

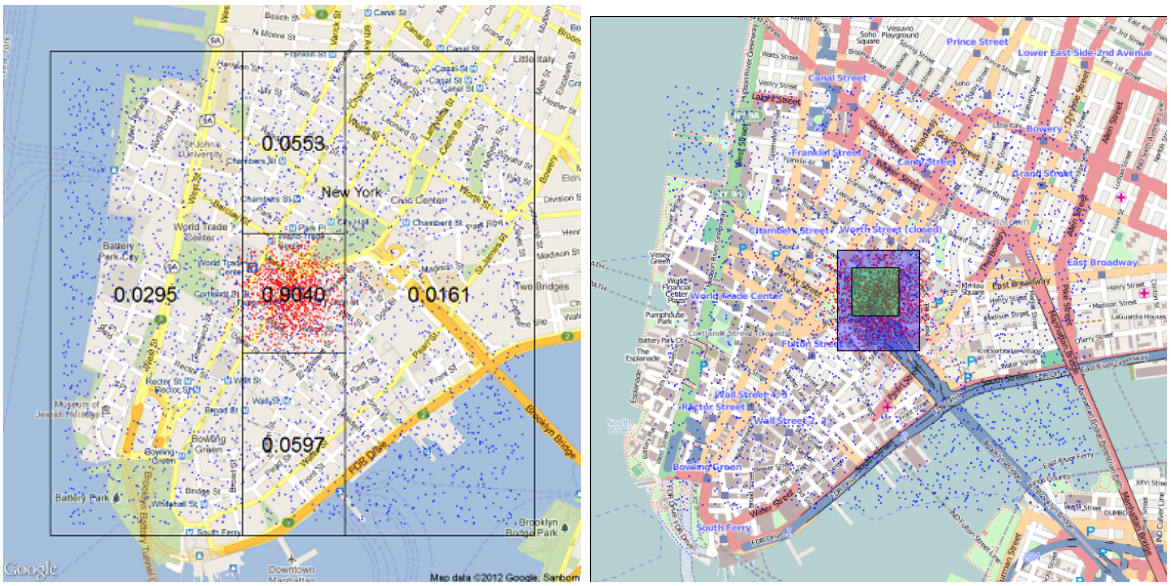

Figure 8: Left: A cluster found by a classification tree visualized on a Google map tile. The numeric labels indicate the fraction of the positive class labels found in the respective rectangle (Figure 6 in Loecher 2012). Right: A cluster found with the patient rule induction method (PRIM), this time visualized on an OpenStreetMap tile (Figure 7 in Loecher 2012) .

containing all of the data. The box is compressed along one face by a small amount, and the observations then falling outside the box are peeled off. After the top-down sequence is computed, PRIM reverses the process, expanding along any edge, if such an expansion increases the box mean. The result of these steps is a sequence of boxes, with different numbers of observation in each box. Cross-validation, combined with the judgment of the data analyst, can be used to choose the optimal box size. We found this algorithm to be naturally tailored to spatial cluster detection and applied it (R package **prim**, Duong 2014) to the same artificial data described above. Two of the resulting boxes found are visualized in Figure 8 right where – to illustrate a possible choice in map style – we chose a map tile obtained from the OpenStreetMap server instead of Google. Both resources can be queried by the `GetMap()` function.

The simulated data are not challenging enough due to their uni modal and symmetric nature. The crime data exhibit multiple clusters at many different scales at various angles. The aggregation at the census block level in the previous section found a very heterogeneous distribution of violent crime rates. What happens if we do not start with a given spatial partition and instead want algorithms such as trees find those regions with high and low densities or rates w.r.t. some baseline? We follow the ideas outlined above and grow trees on various rotations of the data, thereby identifying rectangular "leafs" of high or low incidence of one of the two classes. While a binary recursive tree partitions space exhaustively and in that sense is not naturally suited as a hotspot detector, we depart from this traditional view and simply retain the most "interesting boxes", typically those with an average rate above a chosen threshold. Results are shown in Figure 9 where the tree succeeds in finding spatial rectangles with odds ratios varying greatly. At this point of the analysis no statistical

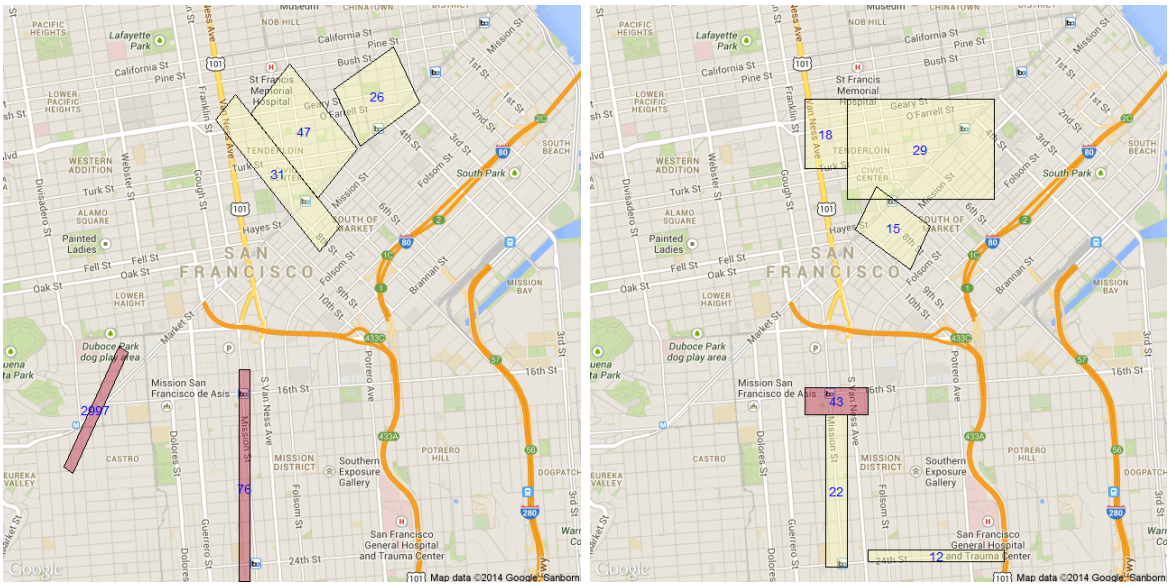

Figure 9: Classification tree (Ripley 2014) fitted to the crime data, where the positive class is given by drug crimes (left) and robbery-related incidents (right). The labels indicate the odds ratios of the respective crimes inside the outlined rectangular leafs. The classification trees are grown on several rotations of the data. The baseline distribution is a uniform background density.

significance accompanies these cells. Note that the user needs to specify a baseline distribution that corresponds to the negative class. For specific crime categories, a natural choice could be "all other categories" which would identify clusters where a much higher/lower rate of that crime occurs *relative to* overall crime. If on the other hand we only wanted to detect over densities, the baseline distribution would be a uniform background incidence which we achieve by artificially adding uniformly distributed points to the data set. Note that future enhancements of this clustering method are planned that directly modify the cost function of the tree algorithm such that deviations from any theoretical baseline distribution can be computed directly, eliminating the need for artificial data augmentations.

5. Extending analyses using trellis structures

Sometimes, especially when we are working with novel data sets, we approach the limits of what a plotting package is capable of. For example, having *a priori* knowledge of our data set we might want to visualize it in a non-conventional fashion that is not supported by the plot package's existing plots but more appropriate to our data set. One of the most powerful elements of R is that it allows users to work with code of multiple levels: Users can generate complex research-grade plots with minimal input using default versions of specialist functions in existing packages, or, with a little extra effort, access and modify the code the existing functions to produce novel visualizations of their own. Here, we provide two examples to demonstrate how particularly amenable the trellis **lattice** framework is to such working practices.

Firstly, we consider the comparison of different surface fitting functions, much like the dis-

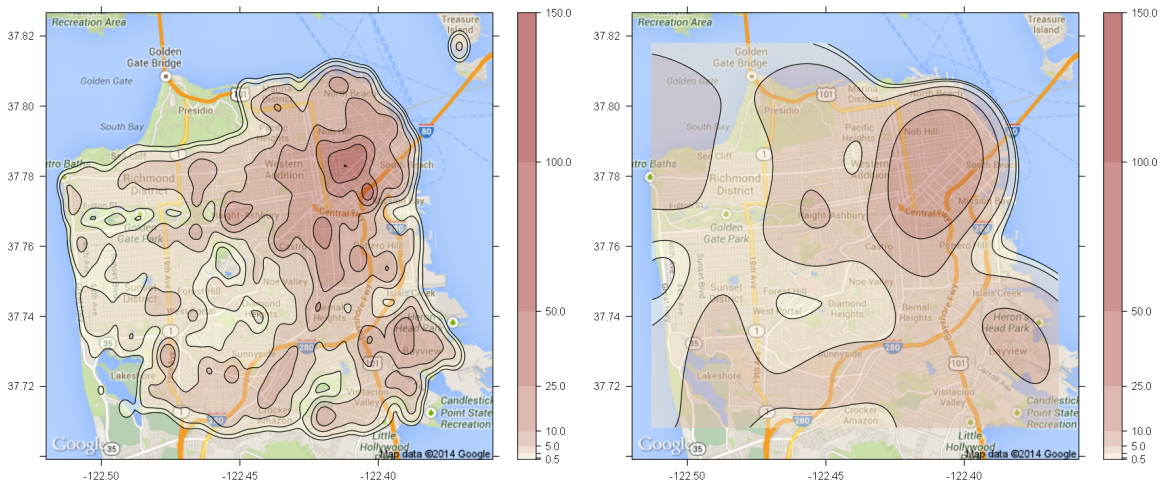

Figure 10: The frequency density of incidents in the San Francisco 2012 data set, estimated using kernel density (left) and GAM (right) surface fits. Note here the GAM outputs are much smoother. However, this effect is most likely in part due to the binning step, an issue that would not be as apparent or easily investigated when working with third-party plot options.

cussion of Trees, GAMs and PRIM in the previous section. We could, for example, decide we would like to compare the performance of one or more of these with surface fitting functions in our own visualizations. However, our plot package might not contain an option to surface fit using, e.g., a GAM. The trellis framework functions to be passed to plots at several levels (e.g., as arguments supplied to panels, or in modified panels or full plot functions). Code in the supporting information for this paper shows how the `loa` `panel.kernelDensity` function was be modified to generate a density surface using a GAM function. As the standard form of the GAM would be `count  s(x, y)` we also incorporate a 'bin and count' step to generate a suitable data set for fitting. The output of this operation is compared with a standard `GoogleMap(..., panel = panel.kernelDensity)` kernel density surface in Figure 10, and both plots are generated within the **lattice**/**loa** framework, so readily amenable to trellis-style conditioning allowing more detailed investigation.

Secondly, we consider the post-plot situation. With most plotting tools the user plots some data, identifies features of interest, but then returns to the pre-plot data to do further work. Because **loa** provides a mechanism for isolating data processing and plotting steps and can retain the results of processing steps, this information can be extracted from the plots themselves. This means, for example, that we can recover the `z` information generated and used by the plots to make the surfaces in Figure 10 and work directly with this information in further analysis. This provides a much more robust link between data visualization and any subsequent data analysis. We can also extend this approach to conditioned data, so, e.g., having plotted surfaces for different types of crimes we can extract and work directly with those different surfaces.

Another common post-plot situation is that we plot data and see (or think we see) clusters that even our most sophisticated data visualization techniques do not seem to be able to isolate. In such cases it is useful to be able to 'snatch' data manually from the plot. **loa** includes a function `getLatLon()` that can be used to manually select points on an existing

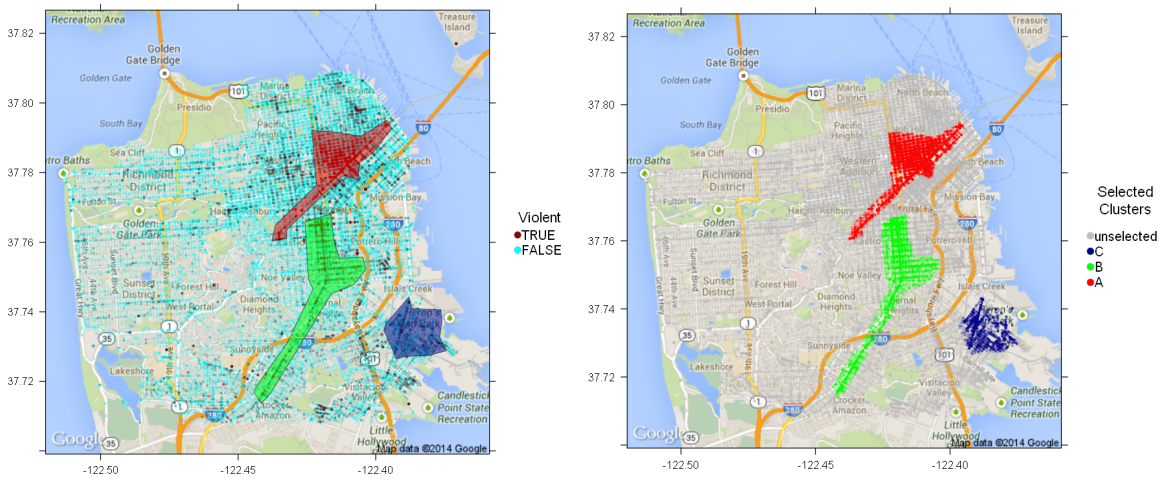

Figure 11: The manual selection of data ranges using a previously generated plot. Left: Three regions manually selected by the authors from a plot similar to Figure 4 using example code supplied in the supporting information for this paper, and shown as red, green and blue polygons. Right: The post-plot manipulation of data within these ranges, here the relatively trivial color-coding of in- and out-of-range data. Note these regions were manual drawn, and used to illustrated options to study features that conventional clustering methods might not identify. The regions are areas of potential interest, that, having been isolated, a user might test for cluster-like characteristics.

loa `GoogleMap()` and recover the latitude and longitude coordinates of these. (Note latitudes and longitudes are locally scaled by the GoogleMap API, and **RgoogleMaps** uses a series of conversion functions, `LatLon2XY()`, `XY2LatLon()`, etc., to work with these. So, `getLatLon` is actually a wrapper for another function, `getXY()`, which gets the raw (x, y) points and the **RgoogleMaps** converter that resets these to latitudes and longitudes.) The supporting information for this paper includes an example that demonstrates how this function can be used with `inout()` in **splancs** (Rowlingson and Diggle 2014) to manually draw a series of polygons on an existing plot and isolate these cases within specific polygons. Such an output could then be used to investigate the validity of such suspected but not independently detected clusters. The use of this function is illustrated in Figure 11.

6. Conclusion

In this paper, we have demonstrated how the contextual information of a map tile can enrich the visualization of spatial data. The packages **RgoogleMaps** and **loa** enable the user to execute GIS type operations all within the same R environment that allows for advanced statistical analysis of the data. Not having to switch environments can be invaluable. While many tools nowadays enable easy map overlays and "mashups", the user typically does not have full flexibility and the graphical prowess that R offers.

We should point out the dangers of mixing colors in the overlay and the map background which can lead to ambiguous graphs. We have taken great care to separate the color spaces of the png map tiles and the information overlaid. The risk of color collisions can be minimized

by choosing gray scale or terrain-style map tiles.
We also feel that annotating clusters or polygons with additional graphics created in R is potentially valuable. The benefits of trellis type conditioning cannot be overstated with data that also have a temporal or categorical dimension. Density maps or clusters typically vary greatly over time or conditioned on other variables; such structural plots can be generated with a few commands in **loa**. In addition, we believe that the supervised methods employed here offer great potential for spatial partitioning and finding over densities with respect to a background that is chosen by the user. Future work will include the refinement of the clustering functions and their integration into the **RgoogleMaps** package as well as further development of functions to manipulate existing plots in **loa**.

References

Almquist ZW (2010). "US Census Spatial and Demographic Data in R: The **UScensus2000** Suite of Packages." *Journal of Statistical Software*, **37**(6), 1–31. URL http://www.jstatsoft.org/v37/i06/.

Becker RA, Cleveland WS, Shyu MJ (1996). "The Visual Design and Control of Trellis Display." *Journal of Computational and Graphical Statistics*, **5**(2), 123–155.

Bivand RS, Pebesma E, Gómez-Rubio V (2013). *Applied Spatial Data Analysis with R*. 2nd edition. Springer-Verlag, New York. URL http://www.asdar-book.org/.

Breiman L, Friedman JH, Olshen RA, Stone CJ (1984). *Classification and Regression Trees*. Wadsworth, California.

Carr D (2014). ***hexbin**: Hexagonal Binning Routines*. R package version 1.27.0, URL http://CRAN.R-project.org/package=hexbin.

City and County of San Francisco (2013). "San Fransico Police Department (SFPD) Crime Incident Data." Calendar-year data can be extracted from https://data.sfgov.org/Public-Safety/SFPD-Reported-Incidents-2003-to-Present/dyj4-n68b/, URL https://data.sfgov.org/.

Cleveland WS (1993). *Visualizing Data*. Hobart Press.

Duong T (2014). ***prim**: Patient Rule Induction Method (PRIM)*. R package version 1.0.15, URL http://CRAN.R-project.org/package=prim.

Gesmann M, de Castillo D (2011). "**googleVis**: Interface between R and the Google Visualisation API." *The R Journal*, **3**(2), 40–44.

Google Developers (2014a). "Google Maps API." URL https://developers.google.com/maps/terms/.

Google Developers (2014b). "Google Maps Image APIs." URL http://code.google.com/apis/maps/documentation/staticmaps/.

Hastie T, Tibshirani R, Friedman J (2009). *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, New York.

Johnson S (2006). *The Ghost Map: The Story of London's Most Terrifying Epidemic and How It Changed Science, Cities and the Modern World*. Riverhead.

Kahle D, Wickham H (2013a). ***ggmap**: A Package for Spatial Visualization with Google Maps and OpenStreetMap*. R package version 2.3, URL http://CRAN.R-project.org/package=ggmap.

Kahle D, Wickham H (2013b). "**ggmap**: Spatial Visualization with **ggplot2**." *The R Journal*, **5**(1), 144–161.

Kemp SE, Jefferis G (2014). ***RANN**: Fast Nearest Neighbour Search*. R package version 2.4.1, URL http://CRAN.R-project.org/package=RANN.

Kilibarda M, Bajat B (2012). "**plotGoogleMaps**: The R-Based Web-Mapping Tool for Thematic Spatial Data." *Geomatica*, **66**(1), 37–49.

Kulldorff M (1997). "A Spatial Scan Statistic." *Communications in Statistics – Theory and Methods*, **65**(6), 1481–1496.

Kulldorff M (2001). "Prospective Time Periodic Geographical Disease Surveillance Using a Scan Statistic." *Journal of the Royal Statistical Society A*, **164**(1), 61–72.

Kulldorff M, Heffernan R, Hartman J, Assuncao R, Farzad FM (2005). "A Space-Time Permutation Scan Statistic for Disease Outbreak Detection." *PLoS Medicine*, **2**(3), e59.

Kulldorff M, Information Management Services Inc (2014). ***SaTScan**: Software for the Spatial, Temporal, and Space-Time Scan Statistic*. URL http://www.SaTScan.org/.

Loecher M (2012). "Identifying and Visualizing Spatiotemporal Clusters on Map Tiles." In *JSM Proceedings, Statistical Graphics Section*, pp. 2854–2863. American Statistical Association.

Loecher M (2014). ***RgoogleMaps**: Overlays on Google Map Tiles in R*. R package version 1.2.0.6, URL http://CRAN.R-project.org/package=RgoogleMaps.

Loecher M, Kumar M (2014). ***RapidPolygonLookup**: Polygon Lookup Using kd Trees*. R package version 0.1, URL http://CRAN.R-project.org/package=RapidPolygonLookup.

Neill DB (2012). "Fast Subset Scan for Spatial Pattern Detection." *Journal of the Royal Statistical Society B*, **74**(2), 337–360.

Pebesma EJ, Bivand RS (2005). "Classes and Methods for Spatial Data in R." *R News*, **5**(2), 9–13. URL http://CRAN.R-project.org/doc/Rnews/.

Potere D (2008). "Horizontal Positional Accuracy of Google Earth's High-Resolution Imagery Archive." *Sensors*, **8**(12), 7973–7981.

R Core Team (2014). "R: A Language and Environment for Statistical Computing." URL http://www.R-project.org/.

Rikken MGJ, Van Rijn RPG (1993). *Soil Pollution with Heavy Metals – An Inquiry into Spatial Variation, Cost of Mapping and the Risk Evaluation of Copper, Cadmium, Lead and Zinc in the Floodplains of the Meuse West of Stein, the Netherlands*. Master's thesis, Physical Geography, Utrecht University.

Ripley B (2014). ***tree**: Classification and Regression Trees*. R package version 1.0-35, URL http://CRAN.R-project.org/package=tree.

Ropkins K (2014). ***loa**: Various Plots, Options and Add-Ins for Use with **lattice***. R package version 0.2.15, URL http://CRAN.R-project.org/package=loa.

Rowlingson B, Diggle P (2014). ***splancs**: Spatial and Space-Time Point Pattern Analysis*. R package version 2.01-36, URL http://CRAN.R-project.org/package=splancs.

Sarkar D (2008). ***lattice:** Multivariate Data Visualization with R*. Springer-Verlag, New York.

Schnute JT, Boers N, Haigh R, Grandin C, Johnson A, Wessel P, Antonio F (2014). ***PBSmapping**: Mapping Fisheries Data and Spatial Analysis Tools*. R package version 2.67.60, URL http://CRAN.R-project.org/package=PBSmapping.

Stevenson M (2014). ***epiR**: An R Package for the Analysis of Epidemiological Data*. R package version 0.9-59, URL http://CRAN.R-project.org/package=epiR.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. 4th edition. Springer-Verlag, New York.

Wickham H (2009). ***ggplot2**: Elegant Graphics for Data Analysis*. Springer-Verlag, New York.

WinVector (2013). *Modeling Trick: Impact Coding of Categorical Variables with Many Levels*. URL http://www.win-vector.com/blog/2012/07/modeling-trick-impact-coding-of-categorical-variables-with-many-levels/.

Affiliation:

Markus Loecher
Department of Business and Economics
Berlin School of Economics and Law
Badensche Str. 52
10825 Berlin, Germany
E-mail: mloecher@hwr-berlin.de
URL: http://www.hwr-berlin.de/about-us/academic-staff/details/markus-loecher/

Karl Ropkins
Institute for Transport Studies
University of Leeds
Leeds LS2 9JT, United Kingdom
E-mail: K.Ropkins@its.leeds.ac.uk
URL: http://www.its.leeds.ac.uk/people/k.ropkins

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/

Volume 63, Issue 4 *Submitted:* 2013-05-31
January 2015 *Accepted:* 2014-10-06