Identifying and Visualizing Spatiotemporal Clusters on Map Tiles

Markus Löcher*

Abstract

Scoring unusual events in space and time has been an active and important field of research for decades: How do we (i) distinguish normal fluctuations in a stochastic count process from real additive events, (ii) identify spatiotemporal clusters where the event is most strongly pronounced and (iii) how do we efficiently graph these clusters in a map overlay ?

Supervised learning algorithms are proposed as an alternative to the computationally expensive scan statistic. The task can be reduced to detecting over-densities in space relative to a background density. We frame the relative density estimation as a binary classification problem.

In the light of recent advances of embedding map tiles in statistical software via the library RgoogleMaps we are developing an integrated hotspot visualizer. The goal is to efficiently identify and visualize spatiotemporal clusters in one environment.

Key Words: scan statistic, RgoogleMaps, hotspots, supervised learning, PRIM

1. Motivation

This paper summarizes recent advances in both the **identification** as well as the **visualization** of spatial/spatiotemporal clusters. The latter is implemented via the R package *RgoogleMaps* whereas the computationally demanding task of searching for regions of very high or very low count density is expedited by supervised learning techniques such as classification trees. The motivation to want to draw on map tiles directly within R is twofold. While the R environment boasts a long history of spatial analysis tools and libraries, the plots have traditionally been created without any spatial context that a map would provide. An example is shown in Figure 1 which shows two different ways of plotting the *meuse* [8] data set. 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 phyisican 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 [6].

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 Map URLs for example are restricted to 2048 characters in length [11] which in practice can be very limiting. Note that the library RgoogleMaps focusses exclusively on static map display within R and does NOT offer any dynamic map mashup capabilities. We refer the interested reader to the fantastic packages *plotGoogleMaps* and *googleVis* instead.

*Berlin School of Economics and Law, mloecher@hwr-berlin.de

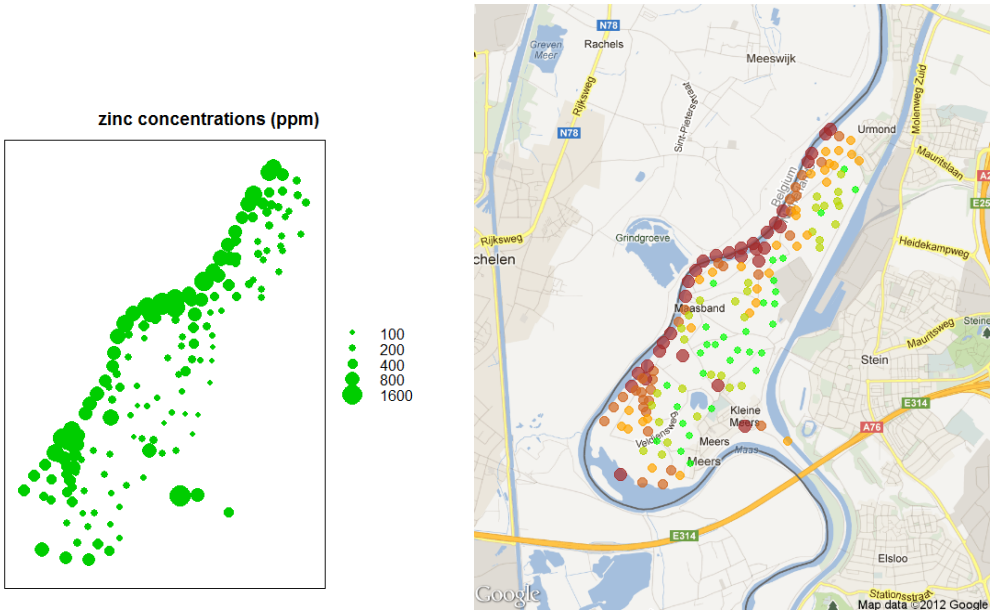

Figure 1: (a) meuse data set visualized by the *bubble()* command in the *sp* library [9], (b) meuse data set visualized by the *bubbleMap()* command in the R library *RgoogleMaps* [10]

2. Integration of maps into R: RgoogleMaps

The Google Static Maps API [11] allows easy download of Google Maps images without requiring JavaScript. The Google Static Map service creates a map based on URL parameters sent through a standard HTTP request and returns the map as an image in various formats. (Note: The Google Static Maps API does NOT require a Maps API key any longer.) 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 [11].
The RgoogleMaps package serves two purposes:

- Provide a comfortable R interface to query the Google server for static maps
- Use the map as a background image to overlay plots within R. This requires proper coordinate scaling as outlined in the Appendix.

We will summarize the three essential commands needed to plot on a map background:

1. *mapSD = GetMap(center=c(32.7073, -117.162), zoom=10, destfile='SDconv.png')*

   This command fetches a map tile which in this example is centered at the San Diego convention center from the Google server and saves it to the local file SDconv.png. It returns a list *mapSD* containing the image as well as all necessary paramaters which are necessary to properly scale coordinates. The resulting map is shown in the left Figure 2.

2. *PlotOnStaticMap(mapSD, lat=myTrails$lon, lon=myTrails$lon, col=myTrails$col, cex = myTrails$cex)*

   This command plots the points specified as a map overlay. 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. The resulting map is shown in the right Figure 2.

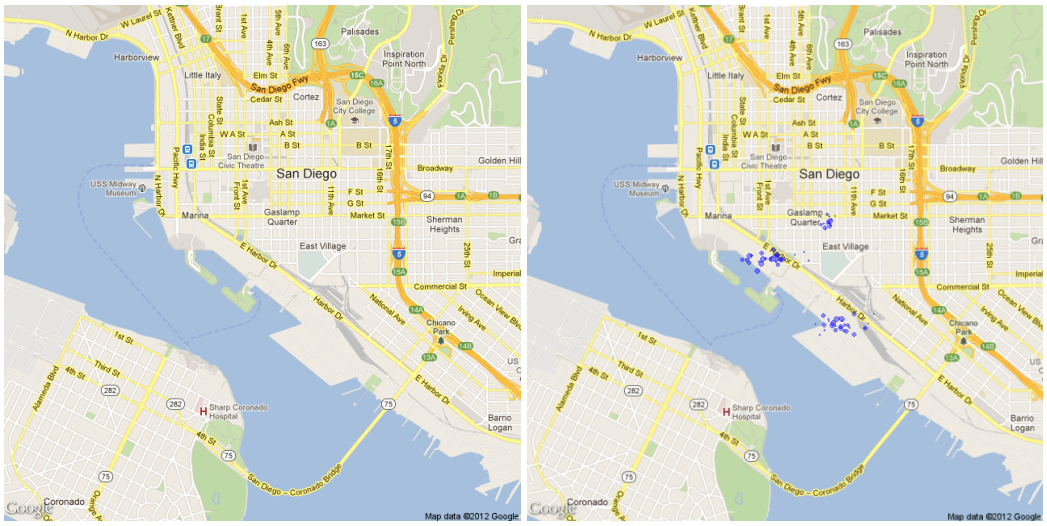

Figure 2: Left: map tile png downloaded by the command mapSD=GetMap(...). Right: plot created within R by the function PlotOnStaticMap(mapSD,...). See text for details.

3. *PlotPolysOnStaticMap(map, shp, lwd=.5, col = shp[,'col']);*

   This function plots/overlays polygons on a map. Typically, the polygons originate from a shapefile. Figure 3 was actually created by a close cousin of PlotPolysOnStaticMap, namely the function ColorMap() which plots Levels of a variable in a colour-coded map. The main difference is that ColorMap can handle spatial objects as defined in the *sp* library directly.

```
bb = qbbox(lat = pennLC$geo[,"y"], lon = pennLC$geo[,"x"])
mapPennLC <- GetMap.bbox(bb$lonR, bb$latR, destfile = "pennLC.png")
ColorMap(100*pennLC$smoking[,2], mapPennLC, pennLC$spatial.polygon,
add = FALSE,alpha = 0.35, log = TRUE, location = "topleft")
```

In closing this section we would like to point out that despite its name the RgoogleMaps library is not restricted to Google map tiles; the function *GetMap.OSM()* queries the *OpenStreetMaps* server for maps, an example of which is shown in Figure 7.

3. 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 [3, 4, 5] and its associated public domain software SatScan^{TM} [12] are widely used for the detection and evaluation of disease clusters. We need to distinguish between two rather different situations:

3.1 Aggregated Counts

In this spatial surveillance setting, each day we have data collected for a set of discrete spatial locations s_i and want to identify "true" clusters of elevated or reduced "activitiy". This task can be broken down into two parts: first figuring out what we expect to see, and then determining which regions deviate significantly from our expectations. For example, in the application of disease surveillance, we examine the spatial distribution of disease cases (or some related quantity, such as the number of emergency department visits or over-the-counter drug sales of a specific type), and our goal is to determine whether any

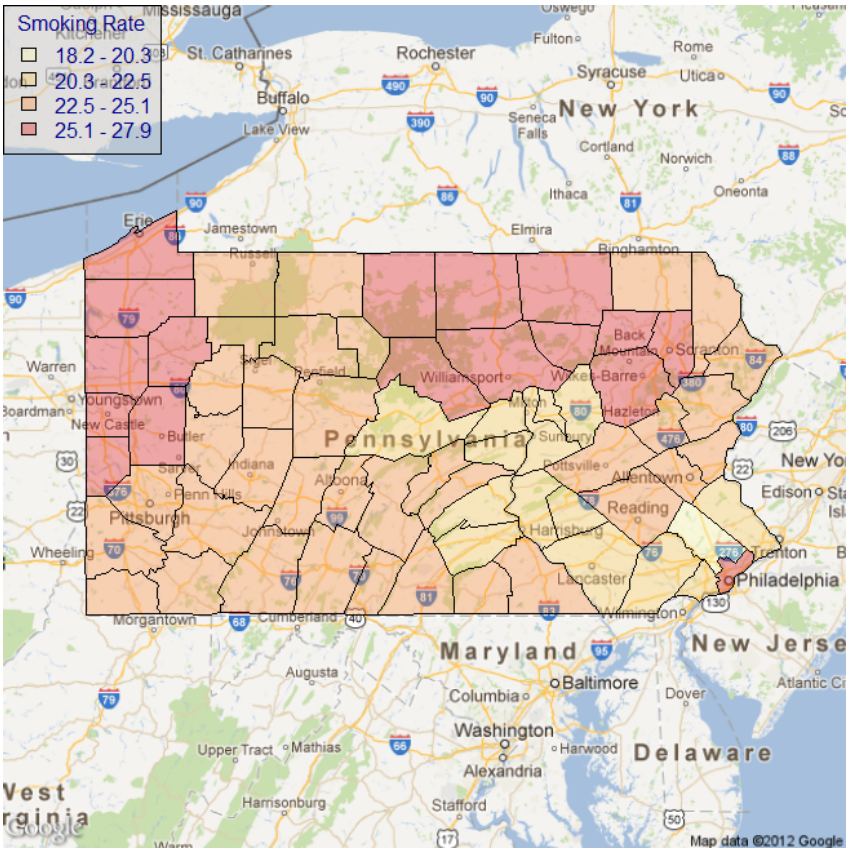

Figure 3: Smoking rates in Pennsylvania as recorded in the *pennLC* data set which contains lung cancer and smoking data from the Pennsylvania Department of Health website.

regions have sufficiently high case counts to be indicative of an emerging disease epidemic in that area. Thus we first infer the expected case count for each spatial location (e.g. zip code), typically based on historical data (though simpler approaches, such as assuming that the number of cases is proportional to census population, can also be used). Then the next step is to determine which (if any) regions have significantly more cases than expected.

In more detail: For each location s_i, we have a count c_i (e.g. number of disease cases), and an underlying baseline b_i. The baseline may correspond to the underlying population at risk, or may be an estimate of the expected value of the count (e.g. derived from the time series of previous count data). Our goal, then, is to find if there is any spatial region S (set of contiguous locations s_i) for which the counts are significantly higher than expected, given the baselines. More formally, we

1. Choose models of the data under H_0 (the null hypothesis of no clusters) and $H_1(S)$ (the alternative hypothesis assuming a cluster in region S).

2. Define a score function $F(S)$ which aims to separate the decision regions between H_0 and H_1 based on aggregate counts alone. The original work by Kulldorff uses the likelihood ratio which simplifies greatly under the Poisson model:

$$F(S) = \frac{Pr(data|H_1(S))}{Pr(data|H_0(S))} = \left(\frac{C_{in}}{B_{in}}\right)^{C_{in}} \cdot \left(\frac{C_{out}}{B_{out}}\right)^{C_{out}} \cdot \left(\frac{C_{all}}{B_{all}}\right)^{-C_{all}} \quad (1)$$

where ”in”, ”out”, and ”all” represent the aggregates of counts and baselines for s_i inside region S, for s_i outside region S, and for all s_i respectively.

3. Find the "most interesting" regions, i.e. those regions S with the highest values of $F(S)$.

4. Perform significance testing.

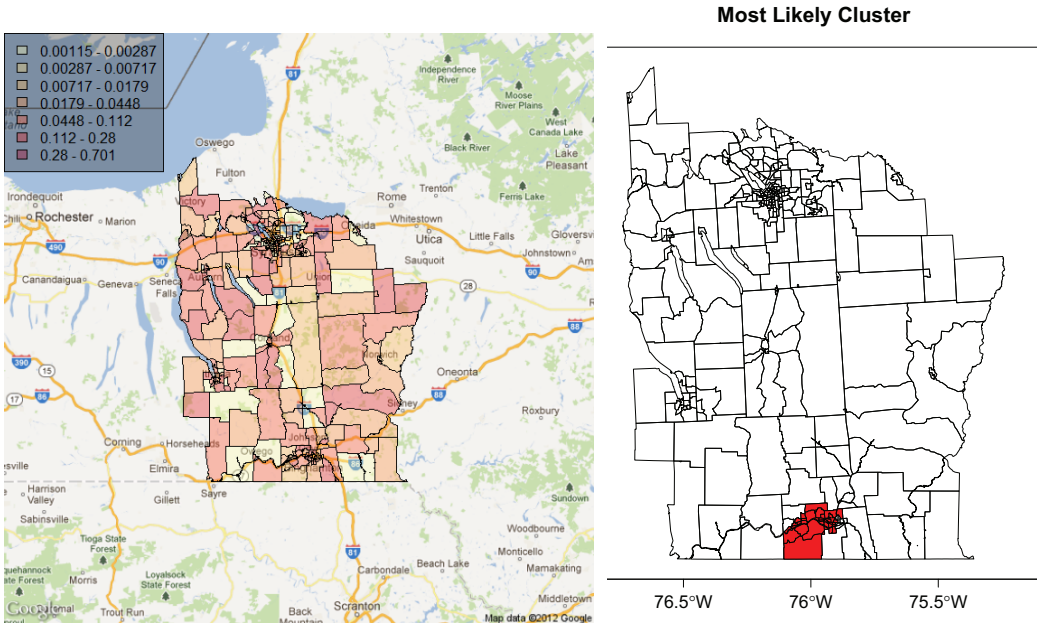

Figure 4: The data set *NYleukemia* contains census tract level ($n = 277$) leukemia data for the 8 counties in upstate New York from $1978 - 1982$, paired with population data from the 1980 census. Left: leukemia incidence plotted with *RgoogleMaps*. Right: clusters found and visualized using the package *SpatialEpi*.

Step 3 from above is usually conducted as a massive, computationally intense, exhaustive search. The next chapter advocates alternative greedy algorithms such as trees that will find significant clusters with high probability and yet are extremely efficient in their complexity.

3.2 Point Process

The models described in the previous section are based on data observed/aggregated at discrete locations that are considered to be non-random, as defined by a regular or irregular lattice of location points. Those are typically referred to as *discrete scan statistics*[12]. In a continuous scan statistics, observations may be located anywhere within a study area. By aggregating the cases and population to discrete cells we often reduce the algorithmic burden by orders of magnitude without losing too much spatial resolution. However, there is no conceptual hurdle of applying the scan statistic and the supervised techniques described in the next section to the original point process data, i.e. with no aggregation performed.

3.3 Unsupervised as Supervised Learning

Hastie *et al* [2](pp. 594-501) introduced the following idea for transforming the density estimation problem into one of supervised function approximation:

> Let $g(x)$ be the unknown data probability density to be estimated, and $g_0(x)$ be a specified probability density function used for reference. The data set $x_1, x_2, ..., x_N$ is presumed to be an i.i.d. random sample drawn from $g(x)$.

A sample of size N_0 can be drawn from $g_0(x)$ using Monte Carlo methods. Pooling these two data sets, and assigning mass $w = N_0/(N + N_0)$ to those drawn from $g(x)$, and $w_0 = N/(N + N_0)$ to those drawn from $g_0(x)$, results in a random sample drawn from the mixture density $(g(x) + g_0(x))/2$. If one assigns the value $Y = 1$ to each sample point drawn from $g(x)$ and $Y = 0$ those drawn from $g_0(x)$, then

$$E(Y|x) = \frac{g(x)}{g(x) + g_0(x)} = \frac{g(x)/g_0(x)}{g(x)/g_0(x) + 1}$$

can be estimated by by supervised learning using the combined sample $(x_1, y_1), (x_2, y_2), \ldots, (x_{N+N_0}, y_{N+N_0})$ as training data.

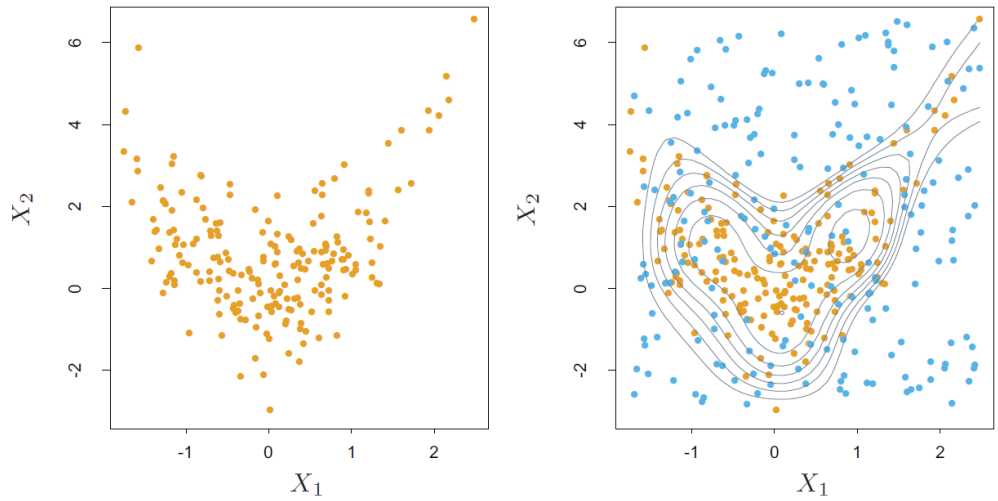

Figure 5: This is Figure 14.3 in [2]. Original caption: "Density estimation via classification. (Left panel:) Training set of 200 data points. (Right panel:) Training set plus 200 reference data points, generated uniformly over the rectangle containing the training data. The training sample was labeled as class 1, and the reference sample class 0, and a semi-parametric logistic regression model was fit to the data. Some contours for g(x) are shown."

An example is shown in Figure 5, see caption for details. We believe that this technique can be gainfully applied to the described epidemiologic setting where cases and population naturally provide two classes. The background population frees us from the need to generate a "fake" reference distribution.

3.4 Detecting simple clusters of overdensity

We chose to apply two very different learning algorithms to a simulated Gaussian bump embedded in uniform background data. The first natural choice are classification trees [1] as implemented by the R library *tree* : 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 6. We also plot the partitions found by the binary recursive algorithm and

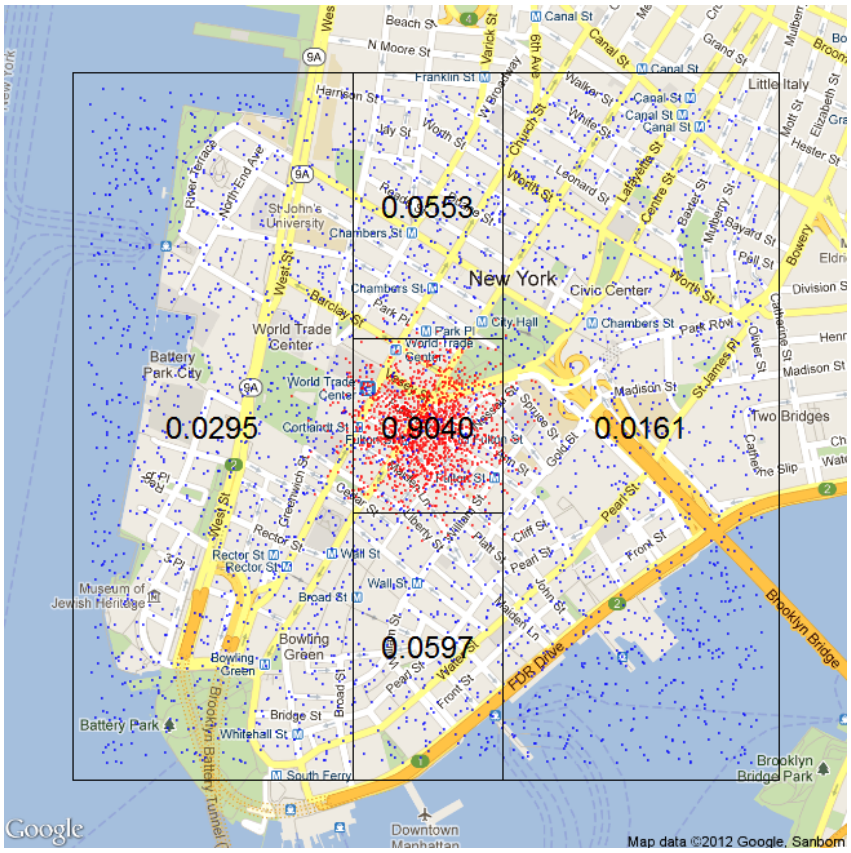

Figure 6: A cluster found by a classification tree visualized on a Google map tile. The numeric labels indicate the fraction of the positive class labels found in the respective rectangle.

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 [12] and even the ”fast scan statistic” [7]. 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 [1, 2]. Minimizing those loss functions is not directly equivalent to maximizing the score function Eq. 1. 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* [2](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 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. The face chosen for compression is the one resulting in the largest box mean, after the compression is performed. Then the process is repeated, stopping when the current box contains some minimum number of

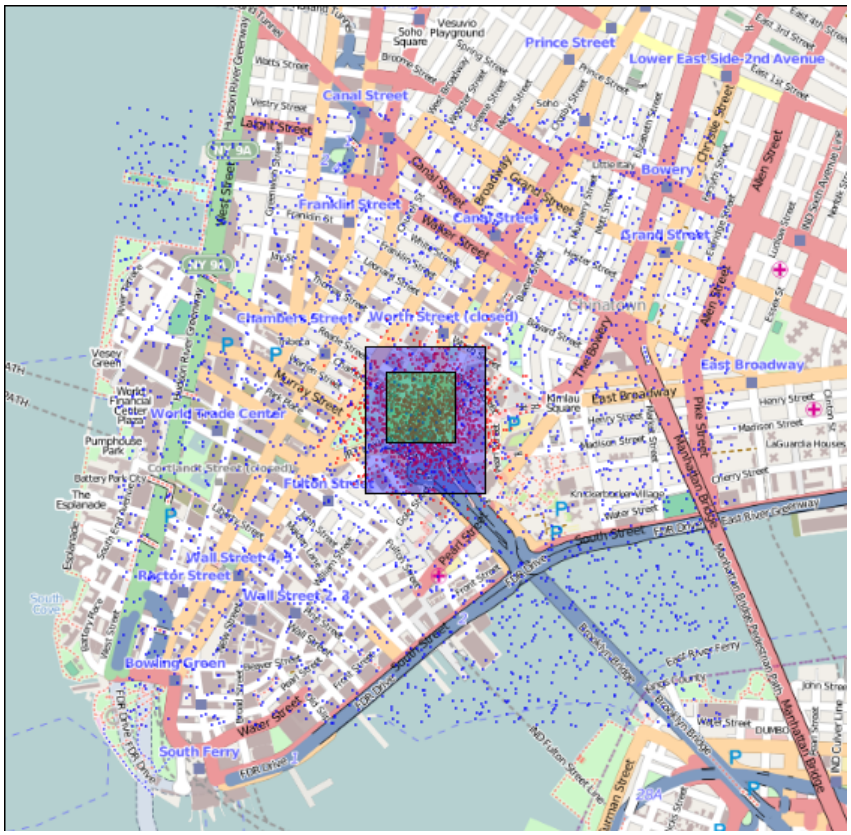

Figure 7: A cluster found with the patient rule induction method (PRIM), this time visualized on an Open Street map tile.

> data points.After the top-down sequence is computed, PRIM reverses the process, expanding along any edge, if such an expansion increases the box mean. This is called pasting. Since the top-down procedure is greedy at each step, such an expansion is often possible. 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, is used to choose the optimal box size.

The authors speculate that "the ability of PRIM to be more patient should help the top-down greedy algorithm find a better solution". We found this algorithm to be naturally tailored to spatial cluster detection and applied it (R package *prim*) to the same artificial data described above. The resulting two boxes found are visualized in Figure 7 where we chose a map tile obtained from the *OpenStreetMaps* server instead of Google.

4. Outlook

The first part of this paper describes a mature and established software package while the ideas put forward in the second part are still in a very early phase. Many open questions remain whether supervised methods could successfully replace or augment the existing scan statistic hotspot search algorithms. The tradeoffs involved seem characteristic of the general "exhaustive versus greedy" algorithmic choice. At this point we have not performed any significance testing or benchmarking which is a natural next step. An often overlooked fact is that randomization can typically be done once "offline" and not repeated for each

new arrangement of cases! That is because we randomly create a large number R of replica grids by sampling under the null hypothesis, given our maximum likelihood parameter estimates for the null. For example, for the Poisson model based approach mentioned above: once we have found the highest scoring region S^* and its score $F^* = F(S^*)$ we generate counts independently from $c_i \sim \text{Poisson}(q_{all} \cdot b_i)$ using the maximum likelihood estimate $q_{all} = C_{all}/B_{all}$. We then find the highest scoring region and its score for each replica grid: the p-value of S^* is $(R_{beat}+1)/(R+1)$, where R_{beat} is the number of replicas with F* higher than the original grid. If this p-value is less than some threshold (e.g. 0.05), we can conclude that the discovered region is unlikely to have occurred by chance, and is thus a significant spatial cluster; we can then examine secondary clusters. Otherwise, no significant clusters exist. Hence, as long as the background populations b_i stay the same, we do **not** have to repeat the randomization for new case distributions c_i ! That is especially important for streaming data settings where decisions have to be made rapidly but offline time is cheap. In that sense the computational burden of the scan statistic for repeated geographic surveiilance is often severely ($R \sim 1000$) overstated in the literature. (Only the isuse of multiple testing is amplified for repeated scans.)
In closing, we would like to point out that both classification trees and the PRIM algorithm can very easily be extended to spatiotemporal data. Adding a temporal dimension is gracefully handled by these types of partitioning methods.

5. Appendix: MapMath Details

We measure latitude and longitude in degrees but assume that the trigonometric functions used below expect units to be in radians.
With $\tilde{lat} = \pi \cdot lat/180$, the transformation from lat/lon to pixels is given by:

$$\tilde{Y} = \frac{1}{2\pi} \log \left(\frac{1 + \sin(\tilde{lat})}{1 - \sin(\tilde{lat})} \right) \tag{2}$$

$$Y = 2^{zoom-1} * (1 - \tilde{Y}), X = 2^{zoom-1} * (\tilde{X} + 1) \tag{3}$$

The integer part of X, Y specifies the tile, whereas the fractional part times 256 is the pixel coordinate within the Tile itself:

$$x = 256 * (X - \lfloor X), y = 256 * (Y - \lfloor Y)$$

Inverting these relationships is rather straightforward. Eq. (2) leads to

$$\tilde{lat} = 2\pi n \pm \sin^{-1} \left(\frac{\exp 2\pi\tilde{Y} - 1}{\exp 2\pi\tilde{Y} + 1} \right) + \pi, n \in \mathbb{Z} \tag{4}$$

whereas inverting Eqs. (3) gives

$$\tilde{Y} = 1 - Y/2^{zoom-1}, \tilde{X} = X/2^{zoom-1} - 1$$

For longitude, the inverse mapping is much simpler: Since $\tilde{X} = lon/180$, we get

$$lon = 180 \cdot \left(X/2^{zoom-1} - 1 \right).$$

As a final remark, note that while latitude increases heading north, Google maps Y coordinates move opposite, i.e. increase southward.

References

[1] Leo Breiman, *Classification and regression trees*, Wadsworth International Group, Belmont, Calif., 1984 (English).

[2] Trevor Hastie, Robert Tibshirani, and J. H Friedman, *The elements of statistical learning data mining, inference, and prediction*, Springer, New York, 2009 (English).

[3] Martin Kulldorff, *A spatial scan statistic*, Communications in Statistics - Theory and Methods **26** (1997), no. 6, 1481–1496.

[4] Martin Kulldorff, *Prospective time periodic geographical disease surveillance using a scan statistic*, Journal of the Royal Statistical Society: Series A (Statistics in Society) **164** (2001), no. 1, 6172 (en).

[5] Martin Kulldorff, Richard Heffernan, Jessica Hartman, Renato Assuno, and Farzad Mostashari, *A SpaceTime permutation scan statistic for disease outbreak detection*, PLoS Med **2** (2005), no. 3, e59.

[6] Johnson, Steven (2006). The Ghost Map: The Story of London's Most Terrifying Epidemic and How it Changed Science, Cities and the Modern World. Riverhead Books. pp. 195196.

`http://en.wikipedia.org/wiki/1854_Broad_Street_cholera_outbreak`

[7] Daniel B. Neill, *Fast subset scan for spatial pattern detection*, Journal of the Royal Statistical Society: Series B (Statistical Methodology) **74** (2012), no. 2, 337360 (en).

[8] The meuse data set gives locations and topsoil heavy metal concentrations, along with a number of soil and landscape variablesat the observation locations, collected in a flood plain of the river Meuse, near the village of Stein (NL). Heavy metal concentrations are from composite samples of an area of approximately 15 m x 15 m.

[9]
```
library(sp);data(meuse);
coordinates(meuse) <- c("x", "y");
bubble(meuse, "zinc", main = "zinc concentrations (ppm)");
```

[10] Inspired by the function *bubbleGoogleMaps()* from the package *plotGoogleMaps* :

```
library(RgoogleMaps); library(sp); data(meuse);
coordinates(meuse)<- ~ x+y;
proj4string(meuse) <-  CRS('+init=epsg:28992');
m<-bubbleMap(meuse,zcol='zinc'');
```

[11] `http://code.google.com/apis/maps/documentation/staticmaps/`

[12] Software to compute the scan statistic: `http://www.satscan.org/`