Geospatial Point Density

by Paul F. Evangelista and David Beskow

Abstract This paper introduces a spatial point density algorithm designed to be explainable, meaningful, and efficient. Originally designed for military applications, this technique applies to any spatial point process where there is a desire to clearly understand the measurement of density and maintain fidelity of the point locations. Typical spatial density plotting algorithms, such as kernel density estimation, implement some type of smoothing function that often results in a density value that is difficult to interpret. The purpose of the visualization method in this paper is to understand spatial point activity density with precision and meaning. The temporal tendency of the point process as an extension of the point density methodology is also discussed and displayed. Applications include visualization and measurement of any type of spatial point process. Visualization techniques integrate **ggmap** with examples from San Diego crime data.

Introduction

The **pointdensityP** package is an example of a tailored package that was created to visualize and quantify spatiotemporal point data (Evangelista and Beskow, 2018). The innovation in the presented methods lies in the implementation, simplification, and general improvements over other methods. Although designed for military analysis, the presented analysis and visualization methods work for any discrete point process that contains a spatial dimension. The point density method in this paper calculates event frequency intensity, or density, within the neighborhood of a given point, accounting for geospatial projection. Additionally, if the point process includes a temporal dimension, the temporal tendency can also be calculated. The temporal tendency is the average age of events within the neighborhood of each point, providing perspective on density trends over time.

The purpose of any type of map projection is simple—to understand a phenomenon within the context of our physical environment. The impetus of the work within this paper resulted from difficult-to-explain output from common spatial density or "hot spot" visualization techniques. Many of these spatial density approaches use a methodology that would be appropriate for a continuous valued phenomenon but does not apply well to discrete phenomenon. Many of the existing density visualization techniques apply a smoothing function that depicts activity where there should be no activity, essentially spreading the density across the area of interest without spatial specificity. Furthermore, many of the density approximation techniques, such as kernel density estimation, create an output value that is not easily interpreted. All too often the output simply provides a relative comparison, with colors representing values that indicate a higher or lower density without providing meaningful quantification of the measured phenomenon.

Related work

Spatial density measurement or "hot spot" analysis is widespread in geospatial literature. Baddeley et al. provide an authoritative overview of numerous spatial point pattern computational methods, to include density methods using **spatstat** (Baddeley et al., 2015). Within R, **kde2d** is probably the most commonly used spatial density function (Ripley et al., 2015; Venables and Ripley, 2002). This function creates a gaussian kernel density estimate for two-dimensional data. Wand and Jones discuss computing techniques for kernel estimators in Wand and Jones (1995) and Wand (1994), describing the technique of binning data prior to estimating the kernel density. The function **bkde2D** implements Wand's binned kernel density estimation technique (Wand and Ripley, 2014). The binning approach reduces the computing cost significantly. Assuming n data points that will create a density estimate for m points, the direct computation of the kernel estimate will be $O(nm)$. Binning maps n points to G grid points, where $G \ll n$, and reduces complexity to $O(Gm)$.

There are several problems related to kernel-based smoothing when applied to discrete spatio-temporal events. Ease of interpretation and outputs designed for a continuous valued process have already been mentioned. For the binned kernel density estimation, granularity of the mesh also quickly increases computational requirements. Kernel based smoothing calculates and returns density for every point in the mesh. Discrete event spatio-temporal phenomenon, especially social phenomenon, only apply to regions of space where people are active. For example, in the San Diego data analyzed in this paper, there is no need to calculate crime density tens of miles into the Pacific Ocean. However, the square mesh approach used in the kernel smoothing algorithms forces this calculation. The `pointdensity()` algorithm presented in this paper returns values only for event locations, improving clarity on both the location and density of event behavior. The following code creates an example

of binned kernel density estimation, using **bkde2D**, applied to the San Diego crime data (San Diego Regional Data Library, 2015). Figure 1 contains the visual results.

```
SD <- read.table("incidents-5y.csv", sep = ",", header = TRUE)
x  <- cbind(SD$lon, SD$lat)
est <- bkde2D(x, bandwidth = c(.01, .01), gridsize = c(750, 800),
  range.x = list(c(-117.45, -116.66), c(32.52, 33.26)))
BKD_df <- data.frame(lat = rep(est$x2, each = 750), lon = rep(est$x1, 800),
  count = c(est$fhat))
map_base <- qmap(location = "32.9,-117.1", zoom = 10, darken = 0.3)
map_base + stat_contour(bins = 150, geom = "polygon", aes(x = lon, y = lat, z = count,
  fill = ..level..), data = BKD_df, alpha = 0.15)
  + scale_fill_continuous(name = "density", low = "green", high = "red")
```

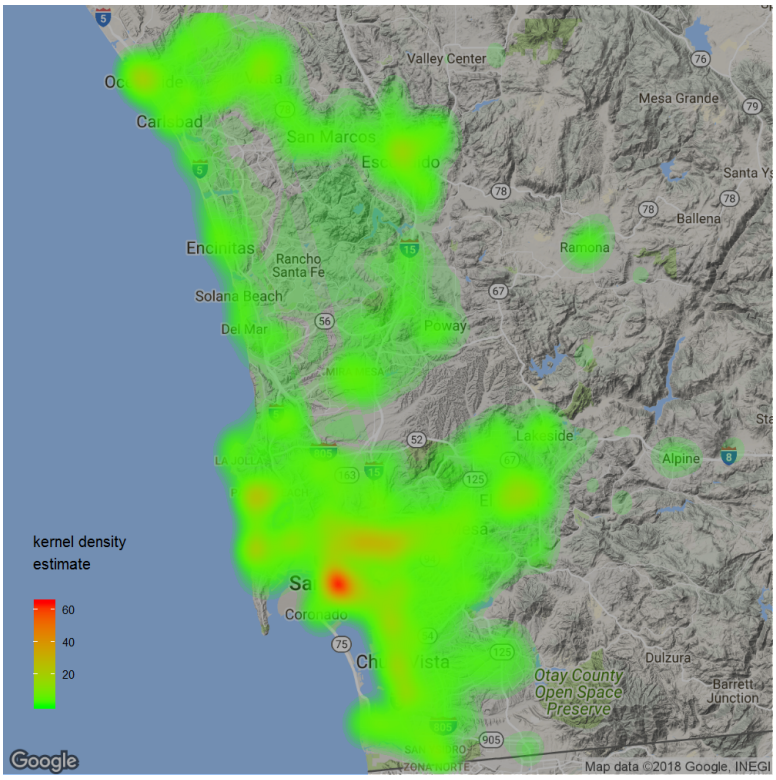

Figure 1: Two-dimensional binned kernel density estimation applied to San Diego crime data.

The method proposed in this paper requires two parameters:

- g, a grid size that represents the fraction of latitude and longitude degree separation in the grid or binning system that will be used for aggregation, and
- r, a positive integer value that represents the radius in grid steps for the neighborhood of interest, which yields a complexity of $O(n(2r)2)$.

Typically, $(2r)^2 \ll G$. As a comparison, Figure 1 resulted from **bkde2D** and compared 800,000 crime locations across a mesh that measured 800 x 750 = 600,000 = G, creating a mesh of points separated by approximately 0.1 km. In comparison, Figure 4 resulted from `pointdensity()` and uses a radius of 10 steps (1 kilometer total) and the same grid size of 0.1 km, comparing 800,000 crime locations each against a local neighborhood of $(2r)^2$ = 400 points. This reduced computational complexity achieved by the `pointdensity()` algorithm largely results from the advantage of assuming and managing an unbounded spatial extent.

Both **bkde2D** and `geom_bin2d()` from **ggplot2** require a bounded spatial extent. Density algorithms with a bounded spatial extent have two common disadvantages: they usually include unnecessary computations, and they always require pre-existing knowledge regarding the spatial extent. While this advantage is not completely analogous to the concept of unbounded data processing, there are similarities. Akidau et al. (2015) describe the fundamental aspects of unbounded data and the need for modern computing methods to accomodate this growing field. Since the `pointdensity()` algorithm processes data without the requirement for a spatial extent, it is possible to efficiently

process data that includes significant spatial separation, a potentially useful trait depending upon the nature of the data under consideration.

The techniques discussed in this paper have been under development for several years and employed in several studies. Previous implementations involved other programming languages (Perl, Python) and leveraged ESRI's *ArcMap* to visualize results (Environmental Systems Research Institute (ESRI), 2018). Although plotting and graphics have long been a strong suit of R, geospatial visualization lagged behind many commercial geospatial information systems until the release of **ggmap** in 2011. **ggmap** is an R package where the authors present methods for plotting geospatial information by leveraging existing online map sources through their APIs and plotting the map as an image in R (Kahle and Wickham, 2013a,b). Similar to *ArcMap*, **ggmap** enables layers of geospatial data to be plotted over a georectified image for the sake of analysis and visualization. The aesthetic appeal of these downloaded images coupled with the capability to overlay sophisticated graphics motivated the creation of the techniques discussed in this paper.

Methodology

Although the claimed merits of the methods in this paper include simplicity and ease of understanding, there are two aspects of this algorithm that may not be immediately apparent without describing in detail. The first involves the method of discretization and geospatial projection used to manage and measure the density. And the second aspect involves the data management necessary to implement this discretization and efficiently calculate the density. Two data management practices will be discussed: the original use of hash data structures and the recent use of matrix-based data structures.

The pointdensity() algorithm

The `pointdensity()` algorithm applies elementary geometry with a geospatial projection in order to build the point density measurements. Figure 2 shows how the neighborhood radius, r, is projected across a square grid. Geographers refer to this as a cylindrical or mercator projection where longitudinal lines appear to remain equidistant. The mercator projection has been used for the sake of illustration, however the algorithm uses a spherical projection and great circle distances to achieve maximal accuracy. As previously mentioned, the algorithm applies a simple binning rule that significantly reduces computations. Density is only collected at the intersection of the grid lines shown in the figure, referred to as grid points. Assume that the center of the circle represents the grid point closest to an event or binned collection of events. The density of every grid point within r will increase by the amount of density associated with the center grid point. The algorithm iterates north and south within r along the latitudes of the grid and calculates the tolerance, t, of longitudinal grid lines that are within r. The spherical Pythagorean theorem is used to find t, providing an accurate binning tolerance irrespective of location on the globe (Veljan, 2018). This binning tolerance ensures that only grid points within r increase in density.

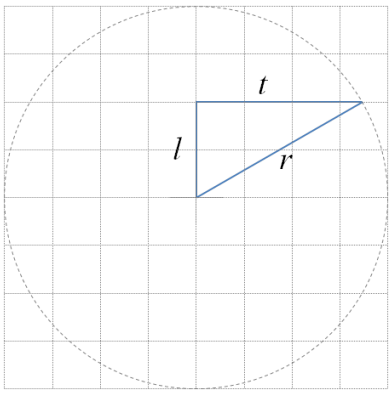

Figure 2: Calculating the longitudinal tolerance.

Figure 3 shows a toy example of the point density method examining only three points. This image shows points A, B, and C, and the radius is four grid steps. Assume that these three points are near the equator and the mercator projection is a reasonable representation of true distance to scale. This figure shows the final density count for every point within the neighborhood radius of the three events.

The original hash-based data structure approach is shown with pseudo-code in algorithm 1. The interested reader can find the implementation of the hash-based algorithm in **pointdensityP** version

Store n points and let lat_i, lon_i, and $date_i$ represent the latitude, longitude, and date of each point, respectively. Let g represent the grid size measured in degrees latitude, and r represent the radius measured in grid steps. For each lat_i and lon_i, round each to the nearest grid point, and store the rounded points as $tlat_i$ and $tlon_i$. Set $m = 0$.
for $i = 1$ to n **do**
  if bin_density_hash$(tlat_i, tlon_i)$ exists **then**
    bin_density_hash$(tlat_i, tlon_i) + +$
    bin_temporal_hash$(tlat_i, tlon_i) += date_i$
  else
    bin_density_hash$(tlat_i, tlon_i) = 1$
    bin_temporal_hash$(tlat_i, tlon_i) = date_i$
    active_grid_hash$(m) = (tlat_i, tlon_i)$
    $m + +$
  end if
end for
for $j = 1$ to m **do**
  retrieve $tlat_j$ and $tlon_j$ from active_grid_hash(j)
  for $lat_t = tlat_j - rg$ to $tlat_j + rg$ **do**
    $t = \arccos(\cos(rg)/\cos(lat_t - tlat_j))/g$
    round t to the nearest integer
    $t = t * g$
    for $lon_t = tlon_j - t$ to $tlon_j + t$ **do**
      density_hash $(lat_t, lon_t) + +$
      temporal_hash$(lat_t, lon_t) +=$ temporal_hash$(tlat_j, tlon_j)$
      $lon_t = lon_t + g$
    end for
    $lat_t = lat_t + g$
  end for
end for
for $i = 1$ to n **do**
  round $(lat_t = lat_i/g)$ to the nearest integer
  $lat_t = lat_t * g$
  round $(lon_t = lon_i/g)$ to the nearest integer
  $lon_t = lon_t * g$
  temporal_hash$(lat_t, lon_t) =$ temporal_hash$(lat_t, lon_t)/$density_hash(lat_t, lon_t)
  print lat_i, lon_i, density_hash(lat_t, lon_t), temporal_hash(lat_t, lon_t)
end for

Algorithm 1: Hash-based point density algorithm

0.2.1. Discussing the original use of hash data structures for the `pointdensity()` algorithm provides two insights: the hash-based approach provides an accessible explanation of the `pointdensity()` computational approach; and secondly, a comparison of the hash-based method to the matrix-based method illustrates a significant difference in computing time when using hash-based data retrieval compared to matrix-based data retrieval. Hash data structures, or associative arrays, are ubiquitous in programming, but considered somewhat of a late arrival to R from a programming standpoint. However, many have pointed out that R's native environment data structures are inherently a hash data structure and can be leveraged as such. Brown (2013) wrote an R package specifically designed to "fully implement" hash data structures in R. The use of hash data structures for the `pointdensity()` algorithm reduced unneeded computations and stored geographic data only within the neighborhood radius of event points. While the use of hash data structures improved the computational complexity from $O(Gm)$ to $O(n(2r)^2)$, the use of matrices and carefully indexed data proved to be an even faster solution. Both computing methods are explained in this paper.

Use of the hash-based data structure achieved desired results, reducing computational complexity while returning an adequate approximation of density. However, the speed of the computation was not satisfactory for larger datasets. While retaining the binned density concept, the algorithm was reduced to one loop of length $= n$ event points using matrix and array facilities. This latter approach will be referred to as the matrix-based approach. The implementation of the matrix-based approach can be found in **pointdensityP** version 0.3.4.

The general process of the `pointdensity()` algorithm reduces a list of n points to a list of m grid

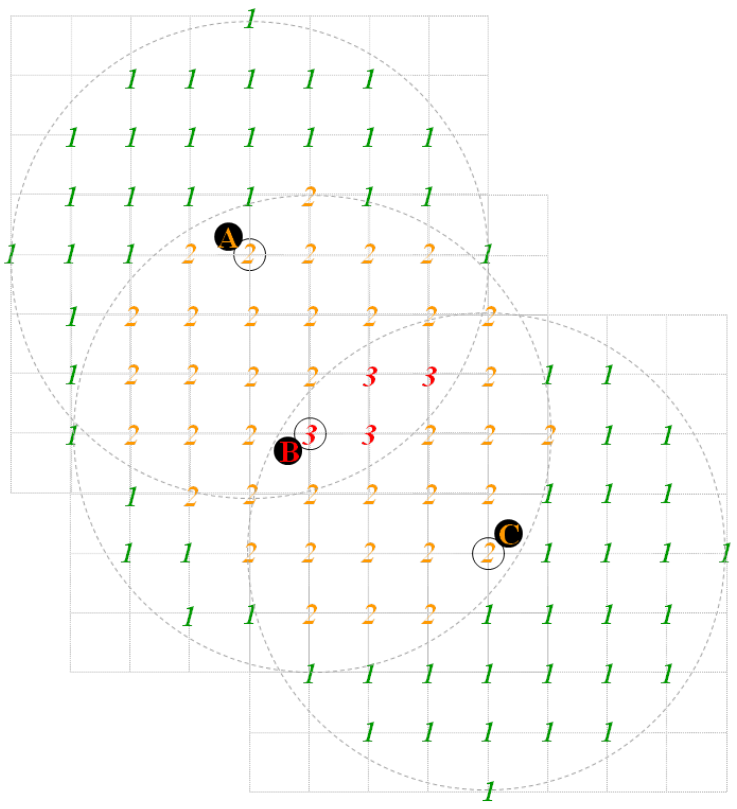

Figure 3: Visualizing the point density algorithm.

points on a mesh. The calculation of density and temporal tendency remains relative to the mesh. Once the overall `pointdensity()` algorithm is complete, it is necessary to assign the density and temporal tendency back to the original list of data. The hash-based algorithm creates intuitive data structures and data retrieval to relate the list of n points to the list of m grid points. Unfortunately, the retrieval of information from the hash tables proved increasingly time consuming as the datasets grew. In order to overcome this problem while continuing to manage the requirement to relate n original points to m grid points, a careful sort and index process, using matrix-based facilities and the **data.table** package, replaced the hash-based storage and retrieval method (Dowle and Srinivasan, 2017). A brief explanation of the improved computing approach follows.

The original list of n points, sorted by their latitude and longitude, reduces to m points once every point is rounded to the nearest point on the mesh. All of the aggregation and sorting leverages functions from the **data.table** package. For each of the m points on the mesh, an initial density of one or greater exists. This initial density is retained through the algorithm. After calculating the final density for each point, this initial density is used to re-expand the list to the original size of n points, in the original order, to return the final density count and temporal tendency.

Each of the m points processes through a density calculation function. The function to calculate the local density surrounding an event point is named `calc_density()` in the **pointdensityP** package. The function starts with enumeration of all latitudes within the radius of the event point, annotated as `lati` and `loni`. These latitudes are stored in a vector:

```
lat.vec <- seq(lati - radius * grid_size, lati + radius * grid_size, grid_size)
```

The variable `radius` represents the radius measured in grid steps, and `grid_size` represents the size of the grid measured in degrees latitude. `lati` is the latitude of the grid point located closest to the event point. `lat.vec` is now an array of each latitude contained in the neighborhood. For each latitude, there is a longitudinal distance that represents the tolerance of the radius. This tolerance is computed using the spherical Pythagorean theorem, as previously discussed. `lat.vec.t` will contain the longitudinal tolerance for each latitude:

```
lat.vec.t <- acos(cos(radius * grid_size) / cos(lat.vec - lati))
lat.vec.t <- lat.vec.t / cos(lat.vec * 2 * pi / 360)
lat.vec.t <- round(lat.vec.t / grid_size, 0) * grid_size
```

The final line of code above rounds the tolerance to the nearest longitude grid line on the mesh.

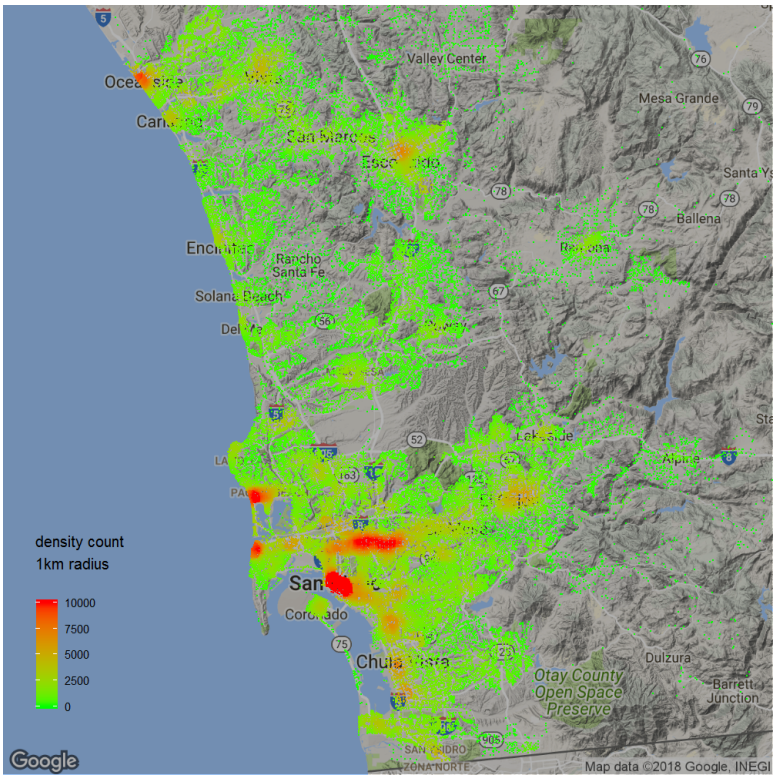

Figure 4: Point density visualization of San Diego crime.

The calculation of `lon.vec` occurs in the same manner as `lat.vec`, which is then further expanded to a matrix, `lon.mat`. This matrix contains the longitudinal value of every point within a square mesh that includes `(2 * radius + 1)^2` points, with the event point at the center. After subtracting `lati` from this matrix, the absolute value of the matrix then contains the distance between the longitude of the event point and the longitude of every point in the mesh. The final subtraction of `lat.vec.t` then returns a matrix that indicates all points outside of the radius as a negative value. After setting all negative elements to zero and all positive values to 1, the Hadamard product of this binary matrix and the original matrix of longitudinal values contained in `lon.mat` returns a matrix that contains a longitudinal value for every point within the radius and a value of 0 for every grid point outside of the radius. Combining these non-zero longitudinal values with the latitude vector returns a list of points within the radius of the event grid point. The final step of the local density calculation involves assigning the temporal value, or sum of temporal values if there is more than one event that rounds to the event grid point, and assigning the original density of the event point to every grid point in the neighborhood. These steps distribute the density and the temporal value of the event point across the neighborhood, completing function `calc_density()` and creating a list of grid points prepared for aggregation without processing a for-loop in R.

As each of the m points processes through `calc_density()`, the list of returned points adds to a **data.table** object which is then aggregated, summing both the density and temporal value for each unique grid point in the **data.table** object. After processing each of the m points and aggregating the returned results, every grid point with a positive density exists in the **data.table** object. The m grid points associated with one of the original n points are retained and re-sorted into their original order, and all other points are discarded. The initial density of every grid point is then added as an additional column to the **data.table**, which is then expanded to length n with the following command:

```
nfinal <- final.1[rep(seq(1, nrow(final.1)), final.1$yy_count)]
```

The **data.table** `final.1` contains the grid point location, density, and sum of temporal values. The field `yy_count` contains the number of original points that were associated with the event's grid point location. After this expansion, a sorted list of n events exists that includes the density and temporal sum of every original point. The temporal sum is then divided by the density. The `pointdensity` algorithm finally returns every original point location, a density value, and the temporal tendency of the point.

Figure 4 displays the calculated density for the San Diego crime data (San Diego Regional Data Library, 2015). Visualizing only the points in the location where the crime occurred provides spatial

specificity, and the coloring provides an explainable measure of the density. Figure 4 results from the following commands:

```
SD_density <- pointdensity(df = SD_sub, lat_col = "lat", lon_col = "lon",
  date_col = "date", grid_size = 0.1, radius = 1)
SD_density$count[SD_density$count > 10000] <- 10000
  ## creates discriminating color scale
map_base + geom_point(aes(x = lon, y = lat, colour = count), shape = 16, size = 2,
  data = SD_density) + scale_colour_gradient(low = "green", high = "red")
```

The difference in speed between the hash-based approach and the newer matrix-based approach was significant. The matrix-based approach eliminates the need for information retrieval from large hash tables, the most expensive aspect of the hash-based approach. Table 1 compares **pointdensityP** version 0.2.1 (hash-based) against version 0.3.2 (matrix-based).

data records	10	10^2	10^3	10^4	10^5	10^6
hash-based time (s)	0.06	0.14	1.19	10.60	265.65	857.15
matrix-based time (s)	0.03	0.15	1.19	7.97	33.35	74.88

Table 1: Time comparison between hash-based algorithm and matrix-based algorithm.

Temporal tendency of event data

A byproduct of the original `pointdensity()` algorithm was the concept of computing the temporal tendency for the events in the neighborhood of every event. Given a set of n events with a spatial and temporal dimension, it is possible to calculate an average event date for the entire dataset or any subset of the data. This is an elementary calculation, however when combined with the neighborhood aggregation approach used in the `pointdensity()` algorithm, it is possible to expose previously undetected trends and patterns.

Figure 5 shows a toy example of two time series plots that span an equivalent period. Clearly the plot on the left will have a smaller temporal tendency due to the heavier density in the earlier periods. This is unarguably a simplistic measure for detecting differences in trends, however it has proven to be a reliable technique in practice.

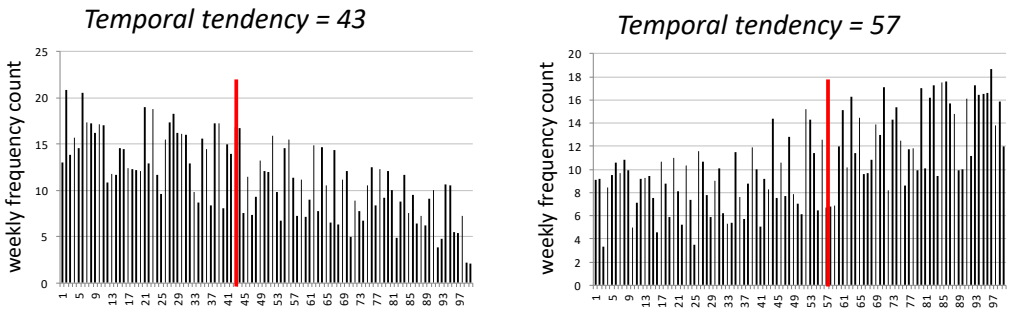

Figure 5: A toy example of the temporal tendency discriminating between an increasing and decreasing trend.

The temporal tendency represents the average age of all events within the neighborhood. This measurement also creates a naive measurement for the relative temporal trend, increasing or decreasing, in the neighborhood. A neighborhood with an increasing trend will have a more recent temporal tendency since a higher density of events occurred more recently. A neighborhood with a decreasing trend will have an older temporal tendency, representing a higher density of events occurring earlier in the time period.

The spatial patterns that emerge from the temporal tendency projection provide the most compelling aspect of the temporal tendency calculation. Some tuning of the color scale is necessary to create visual discrimination. The following lines of code create the underlying temporal tendency plot and histograms shown in Figure 6:

```
SD_temp_tend <- SD_density[SD_density$date_avg > 0]
SD_temp_tend$dateavg[SD_temp_tend$dateavg < 14711] <- 14711
```

```
SD_temp_tend$dateavg[SD_temp_tend$dateavg > 14794] <- 14794
map_base + geom_point(aes(x = lon, y = lat, colour = dateavg), shape = 16, size = 2,
  data = SD_temp_tend) + scale_colour_gradient(low = "green", high = "red")
```

Histograms result from the following:

```
x <- as.Date(SD$date)
hist(x, "weeks", format = "%d %b %y", freq = TRUE)
mid_city <- subset(SD, lat > 32.69 & lon > -117.14 & lat < 32.79 & lon < -117.08)
x <- as.Date(mid_city$date)
hist(x, "weeks", format = "%d %b %y", freq = TRUE)
encinitas <- subset(SD, lat > 33 & lon > -117.32 & lat < 33.09 & lon < -117.27)
x <- as.Date(encinitas$date)
hist(x, "weeks", format = "%d %b %y", freq = TRUE)
```

Figure 6 shows the results of temporal tendency projection for the San Diego crime data. Regions of disparate activity become apparent with this projection, providing an opportunity to detect areas that are "heating up" compared to areas that are "cooling down". The overall San Diego crime dataset shows an aggregate decreasing trend for 2007 - 2012. Box 1, Encinitas, shows a slightly increasing trend. Mid-City is a region with one of the strongest decreasing trends.

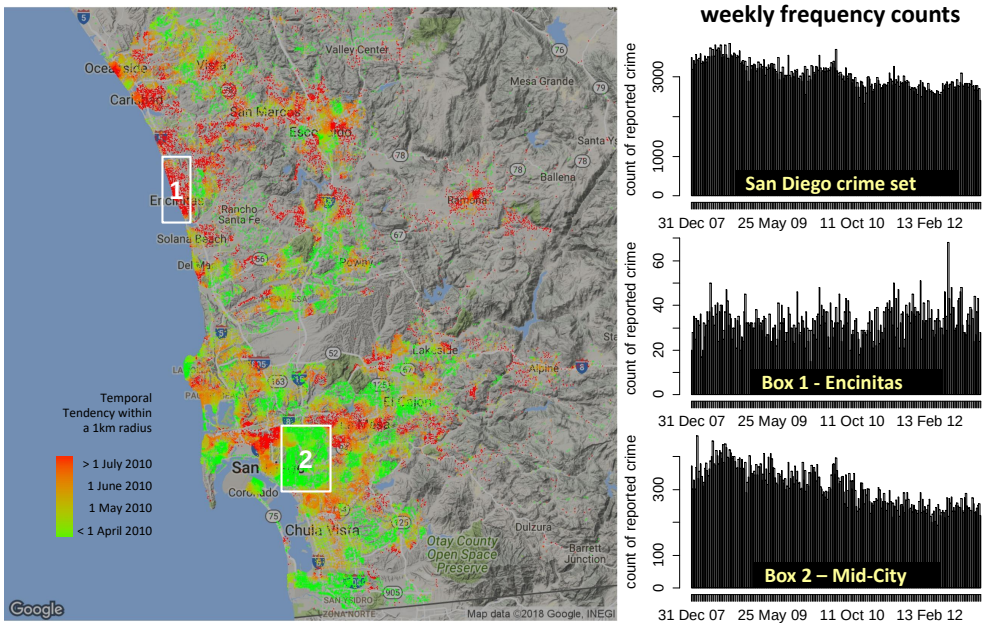

Figure 6: Temporal tendency projection of San Diego crime data.

Simple linear regression is a common method to detect trends. The following code creates a linear model and summary of the model's utility for any of the histograms discussed above:

```
res <- hist(x, "weeks", format = "%d %b %y", freq = TRUE)
xres <- res$breaks[1:(length(res$breaks) - 1)]
summary(lm(res$counts ~ xres))
```

The simple linear regression estimate for the slope coefficient parameter was significant for each of the three regions explored. The map and histograms in Figure 6 highlight the two sub-regions analyzed, Encinitas and Mid-City, as well as the San Diego super-set region that includes all of the data. The trends for each of these regions are apparent from the histograms, and the model utility test results shown in Table 2 for each of these regions shows that there is statistically significant evidence that the slope coefficients are not zero, indicating that a trend exists. This test provides both evidence of the existence of a trend and also quantifies the expected rate of change. The San Diego crime rate in its entirety is dropping by about one crime every two days during this period. Encinitas, lit up in red

on the map, shows a relatively more recent temporal tendency when compared to crime across the entire San Diego region, indicating that the rate of crime change in Encinitas will be increasing at a greater rate when compared to the entire San Diego region. The Encinitas histogram shows evidence of a slight increase, and the model utility test for the slope coefficient provides evidence that the slope is greater than zero. On the opposite end of the spectrum, Mid-City averages a crime rate decrease that drops by 1 crime every 10 days. The green coloring on the map clearly highlights an older temporal tendency in this region, which is reinforced by the histogram and regression analysis.

	Estimate	Std. Error	t statistic	Pr(>|t|)
San Diego Total	-0.5077	0.0204	-24.89	<2e-16
Encinitas	0.0023	0.0008	2.88	0.00435
Mid-City	-0.0958	0.0037	-25.81	<2e-16

Table 2: Simple linear regression slope coefficient estimates for weekly crime frequencies shown in Figure 6.

Realize that the temporal tendency calculation is sensitive in areas with a low density. When visualizing the map-based plot, it is possible to compare regions with minimal density with regions that have a much more significant density. For this reason, it is often advisable to use the temporal tendency visualization alongside the point density visualization and possibly restrict the temporal tendency to areas with a minimum density threshold.

Conclusion

Both the temporal tendency projection and point density projection help illustrate regions of disparate activity. These are regions where event behavior is significantly different, oftentimes warranting investigation into underlying causes. The temporal tendency analysis provides a useful perspective on a commonly elusive dimension of spatio-temporal data: change. The most important insights involving spatio-temporal phenomenon commonly relate to change over time.

Future directions for this work include the calculation, exploration, and visualization of other summary statistics that summarize spatio-temporal behavior. For example, estimates for the variance and skew of the temporal behavior readily extends from the aggregation of the squared and cubed temporal values. Analysis of these measurements could provide better fidelity on trends. Lastly, automating the detection of boundaries of regions of disparate activity could provide significant contributions in identifying cause and effect of spatio-temporal behavior.

Acknowledgments

The authors would like to express appreciation for the thoughtful comments and review of this article by the editors of the R Journal and Ian Irmischer. The opinions expressed in this article are the authors' own and do not reflect the view of the United States Army or the United States government.

Bibliography

T. Akidau, R. Bradshaw, C. Chambers, S. Chernyak, R. J. Fernandez-Moctezuma, R. Lax, S. McVeety, D. Mills, F. Perry, E. Schmidt, and S. Whittle. The dataflow model: A practical approach to balancing correctness, latency, and cost in massive-scale, unbounded, out-of-order data processing. *Proc. VLDB Endow.*, 8(12):1792–1803, 2015. ISSN 2150-8097. URL https://doi.org/10.14778/2824032.2824076. [p348]

A. Baddeley, E. Rubak, and R. Turner. *Spatial Point Patterns: Methodology and Applications with R*. Chapman and Hall/CRC Press, London, 2015. URL https://doi.org/10.1201/b19708. [p347]

C. Brown. *Hash: Full Feature Implementation of Hash/Associated Arrays/Dictionaries*, 2013. URL http://CRAN.R-project.org/package=hash. R package version 2.2.6. [p350]

M. Dowle and A. Srinivasan. *Data.table: Extension of 'data.frame'*, 2017. URL https://CRAN.R-project.org/package=data.table. R package version 1.10.4-3. [p351]

Environmental Systems Research Institute (ESRI). ArcGIS Release 10.6.1, 2018. [p349]

P. Evangelista and D. Beskow. *pointdensityP: Point Density for Geospatial Data*, 2018. URL https://CRAN.R-project.org/package=pointdensityP. R package version 0.3.4. [p347]

D. Kahle and H. Wickham. ggmap: Spatial Visualization with Ggplot2. *The R Journal*, 5(1):144–161, 2013a. [p349]

D. Kahle and H. Wickham. *Ggmap: A Package for Spatial Visualization with Google Maps and OpenStreetMap*, 2013b. URL http://CRAN.R-project.org/package=ggmap. R package version 2.3. [p349]

B. D. Ripley, W. N. Venables, D. M. Bates, K. Hornik, A. Gebhardt, and D. Firth. *MASS: Support Functions and Datasets for Venables and Ripley's MASS*, 2015. URL http://CRAN.R-project.org/package=MASS. R package version 7.3-37. [p347]

San Diego Regional Data Library. San Diego region crime incidents 2007-2013, 2015. URL https://s3.amazonaws.com/s3.sandiegodata.org/repo/clarinova.com/crime-incidents-casnd-7ba4-r3/incidents-5y.csv. Accessed: 2018-03-01. [p348, 352]

D. Veljan. The 2500-Year-Old Pythagorean Theorem. *Mathematics Magazine*, 73 (4):0 259–272, 2018. URL https://doi.org/10.1080/0025570x.2000.11996853. [p349]

W. N. Venables and B. D. Ripley. *Modern Applied Statistics with S*. Springer-Verlag, New York, 4th edition, 2002. [p347]

M. Wand. Fast Computation of Multivariate Kernel Estimators. *Journal of Computational and Graphical Statistics*, 3(4):433–445, 1994. [p347]

M. Wand and B. Ripley. *KernSmooth: Functions for Kernel Smoothing for Wand & Jones (1995)*, 2014. URL http://CRAN.R-project.org/package=KernSmooth. R package version 2.3-13. [p347]

M. P. Wand and M. C. Jones. *Kernel Smoothing*. Monographs on statistics and applied probability. Chapman & Hall/CRC, Boca Raton (Fla.), London, New York, 1995. ISBN 0-412-55270-1. [p347]

Paul F. Evangelista
United States Military Academy
West Point, NY
United States
paul.evangelista@westpoint.edu

David Beskow
United States Military Academy
West Point, NY
United States
david.beskow@westpoint.edu