Kernel-based Visualization of Point Patterns in Python with Application to Tornado Landing Data

Weifeng Hu
Computer Science, University of Michigan *

March 25, 2015

*Mentored by Dr. Stilian Stoev

Abstract

Many data sets come as point patterns of the form (longitude, latitude, time, magnitude). The examples of data sets in this format includes tornado events, origins/destination of internet flows, earthquakes, terrorist attacks and etc. It is difficult to visualize the data with simple plotting. This research project studies and implements non-parametric kernel smoothing in Python as a way of visualizing the intensity of point patterns in space and time. A two-dimensional grid M with size $m_x \times m_y$ is used to store the calculation result for the kernel smoothing of each grid points. The heat-map in Python then uses the grid to plot the resulting images on a map where the resolution is determined by m_x and m_y. The resulting images also depend on a spatial and a temporal smoothing parameters, which control the resolution (smoothness) of the figure. The Python code is applied to visualize over 56,000 tornado landings in the continental U.S. from the period 1950 - 2014. The magnitudes of the tornado are based on Fujita scale.

1 Introduction

Internet traffic flow can be visualized as large point patterns in a plane. The Internet data comes in the format of $(longitude, latitude, payload, time)$. The same data format is also applicable to tornadoes touchdown data, where the payload is the Fujita scale of tornado incidents, earthquake and terrorist attacks. Since the data varies both spatially and temporally, it is difficult to visualize them using simple plot. As a result, this research project implements in Python kernel smoothing, which is a non-parametric statistical methodology that can easily and computationally efficiently visualize large point patterns in space. Kernel smoothing allows visualization and further analysis of spatial-temporal features in point-patterns such as locations, timing and magnitude of the internet traffic.

This research report describes the experimental methods of implementing kernel smoothing in Python programming language. We use 56,000 tornado data from the past five decades and visualize them using Python heat map. In particular we will be focusing on the formulas that are applied to the input data. The output of the Python program will be a matrix containing the results of kernel smoothing and the matrix is plotted on a heat map of the United States based on the values of its entries. Further analysis includes taking the differences of two output images with different temporal bandwidth to observe the change in tornado intensity over different periods of time.

This project also studies how to improve the computational efficiency in Python when kernel smoothing is applied to relatively large data sets and/or

high resolution of the resulting images is required. We implemented an algorithm that creates a smaller two-dimensional grid S with size $n \times n$. For a point on the bigger grid M, we only apply kernel smoothing to grid points that fall within the grid S. Intuitively, the size of grid M plays no role in computation in this algorithm, which allows us to handle larger resolution in relatively efficient way.

2 Kernel Smoothing Methodology

Kernel smoothing is a technique to estimate the real valued function by using its noisy observations. Consider the data set (x_i, y_i, t_i, p_i), where x is the longitude, y is the latitude, t is the time and p is the payload of the network data. In order to visualize the data points in the map, we need to use couple formulas for kernel smoothing. The kernel of the data can be calculated using the formula

$$K(x, y) = \frac{1}{2 \cdot \pi} \cdot e^{\frac{-(x^2+y^2)}{2}} \tag{1}$$

K is the probability density function of the covariate normal distribution with zero mean and variance to be the covariance matrix.

$$\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

After we obtain the formula for kernel, the spatial kernel smoothing can also be obtained using

$$K_h(x, y) = \frac{1}{h^2} \cdot K(\frac{x}{h}, \frac{y}{h}) \tag{2}$$

In formula 2, h is the smoothing parameter called bandwidth. It has strong influences in the result. However, choosing a appropriate bandwidth is relatively difficult.

We also consider temporal kernel smoothing because the data varies with time. Temporal kernel smoothing uses a different bandwidth, denoted as h_t. The formula for temporal kernel smoothing, when given a bandwidth h_t is shown below. Note that temporal kernel smoothing only affects the point after a specific time. If the input time is negative, meaning the data happens before the time, we return the temporal kernel smoothing as zero.

$$T_\lambda(t) = \begin{cases} \lambda \cdot e^{-t\lambda} & t > 0 \\ 0 & t \leq 0 \end{cases} \tag{3}$$

We visualize the data points on a map using Python heat map by first creating a two-by-two grid M. We divide the grid, according to user's input of

number of pixels in x direction m_x and number of pixels in y direction m_y, such that

$$x_k = x_0 + \Delta x \cdot (k + \frac{1}{2}), k = 1, 2, 3 \cdots, m_x - 1 \tag{4}$$

$$y_l = y_0 + \Delta y \cdot (l + \frac{1}{2}), l = 1, 2, 3 \cdots, m_y - 1 \tag{5}$$

Therefore, for every x, y point in the grid M, when given a bandwidth h and data of size N, can be calculated using

$$P_h(x, y, t) = \frac{1}{N} \cdot \sum_{i=1}^{N} p_i \cdot K_h(x - x_i, y - y_i) \cdot T_\lambda(t - t_i) \tag{6}$$

The following figure shows the implementation in Python of the kernel smoothing formulas.

```
# INPUT: longitude, latitude
# EFFECT: calculates the kernel based on longitude and latitude
def kernel (longitude, latitude):
        CONST_e = math.e
        CONST_pi = math.pi
        # kernel = (2Pi)^(-1) * (e)^(-(x^2 + y^2)/2) where x is longitude and y is latitude
        kernel = math.pow(CONST_e, -(math.pow(longitude, 2) + math.pow(latitude, 2))/2) / (2* CONST_pi)
        return kernel
```

3 Algorithm

The algorithm, however, is in order of $m_x \cdot m_y \cdot N$ when implemented using nested loops in Python. The scaling efficiency becomes problematic when the user requires high resolution, i.e. bigger values of m_x and m_y. As a result, a new algorithm is implemented to improve the efficiency of the program. The pseudo-code is described below.

```
1: procedure CALC_INDEX(min, max, position, pixel)
2:     index = int(pixel · (position−min)/(max−min))
3:     return index
4: end procedure
5:
6: procedure S_MATRIX_ENTRIES(k, h, row, col)
7:     entry = 1/(2·π·h²) · e^(−1/h² · ((row−(k+1))² + (col−(k+1))²))
8:     return entry
9: end procedure
10:
11: for i = 0 to 2 · k do
12:     for j = 0 to 2 · k do
13:         S(i, j) = S_MATRIX_ENTRIES(k, bandwidth, i, j)
14:     end for
15: end for
16:
17: S = normalize(S)
18:
19: for i = 0 to number of data points do
20:     X_index = CALC_INDEX(min(x_i), max(x_i), x_i, m_x)
21:     Y_index =CALC_INDEX(min(y_i), max(y_i), y_i, m_y)
22:     m_idx_left = max(0, X_index − k)
23:     m_idx_right = min(X_index + k + 1, m_x)
24:     m_idy_left = max(0, Y_index − k)
25:     m_idy_right = min(Y_index + k + 1, m_y)
26:     s_idx_left = max(0, k − X_index)
27:     s_idx_right = min(k + m_x − X_index, 2 ∗ k + 1)
28:     s_idy_left = max(0, k − Y_index)
29:     s_idy_right = min(k + m_y − Y_index, 2 ∗ k + 1)
30:     x_m = [m_idx_left : m_idx_right]
31:     x_s = [s_idx_left : s_idx_right]
32:     y_m = [m_idy_left : m_idy_right]
33:     y_s = [s_idy_left : s_idy_right]
34:     M[x_m, y_m] += S[x_s, y_s] · p_i · T_λ(t − t_i)
35: end for
```

This algorithm uses a smaller two dimensional grid S of size $s \times s$,to apply kernel smoothing only to the grid points that are close to the data of interest, i.e. those falling within the grid S. The value of k determines the size of grid S. Intuitively the algorithm, when given a data size N, is in order of $k^2 \cdot N$. We can see that it is easy to handle higher resolution map now because the pixels m_x, m_y play no role in computation except memory storage in computers.

Notice that the last part of the pseudo-code presented involves the calcula-

tions to update the specific regions of the matrix. The math involves using max or min to find out which regions of matrix M needs to be updated by the matrix S. We take into consideration of the edge cases where the smaller S matrix may go out of bound of the bigger M matrix when applying kernel smoothing to a region. After we manage to determine which regions need to be updated, we used vectorization in Python that helps improve the speed of the program.

4 Result & Applications

We first apply the algorithm for tornado data, which is similar to internet data because it has touchdown longitude and latitude, time and Fujita scale. We extracted all 56,000 tornado data for the past 50 years. The data is loaded in Python and the heat map is drawn on the map of the United States.

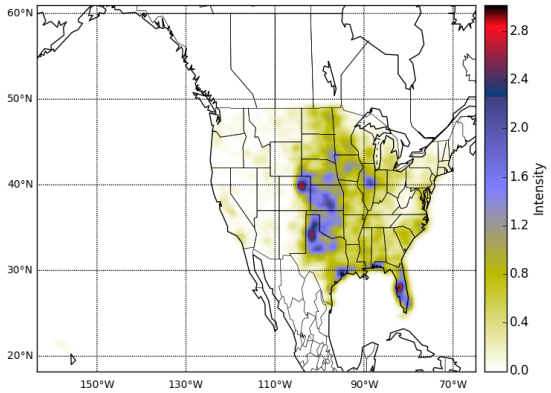
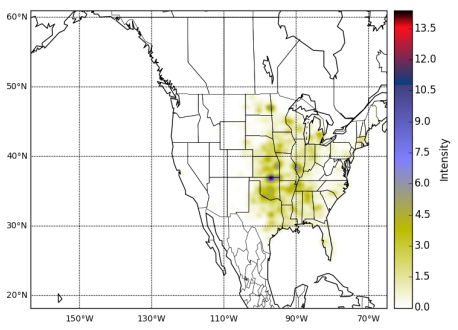


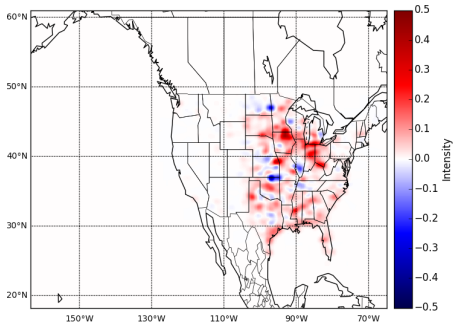
Figure 1: Tornado Touchdown Data Visualized on Heatmap of the United States

We can see the "tornado alley" in the US map from Figure 1. The "tornado alley" is area where tornado is most frequent. The core of "tornado alley" extends from northern Texas, Oklahoma, Kansas and Nebraska. Figure 1 clearly shows that the states with high frequency of tornado are Oklahoma, Texas, Colorado and Kansas, which is very consistent with the areas that are recognized as in the "tornado alley". The values on the colorbar is calculated based on the temporal bandwidth, spatial bandwidth and Fujita scale of the touchdown tornado using the calculation described in Algorithm section.

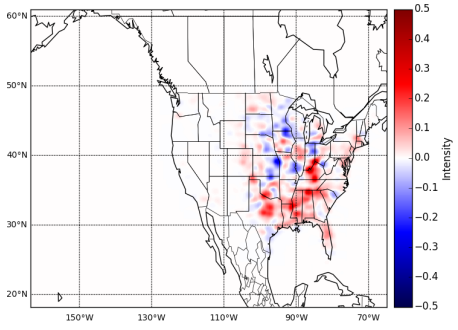
We can also investigate the change in intensity over years. The plots below shows how the intensity of tornado touchdown changes in the United States in a ten-year interval for the past 50 years.



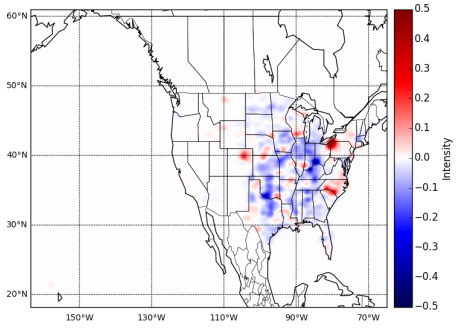
(a) Intensity of Tornado From 1950 to 1960



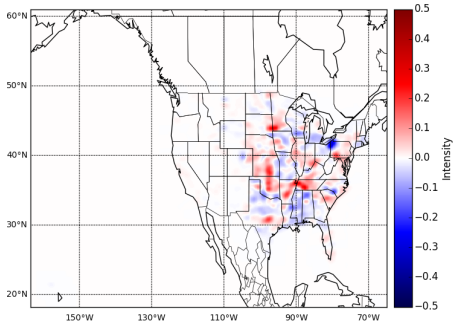
(b) Tornado Intensity Change From 1960 to 1970



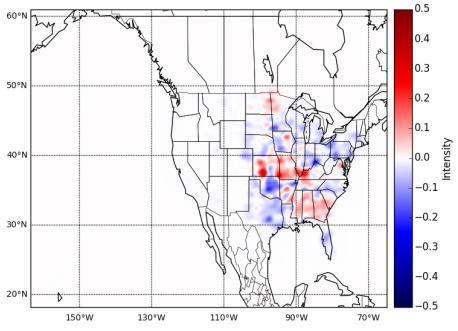
(c) Tornado Intensity Change From 1970 to 1980



(d) Tornado Intensity Change From 1980 to 1990



(e) Tornado Intensity Change From 1990 to 2000



(f) Tornado Intensity Change From 2000 to 2010

It is clear in the plots that for the first ten years(Figure (a)), there are more incidents of tornado touchdowns in states such as Oklahoma and Kansas. In Figure (b), we can see that the intensity of tornado shows a dramatic increase in

the northern part of the country like Wisconsin, Iowa and Illinois. In Figure (c) the intensity of the tornado in the northern part of the United States decreased while the intensity of tornado in southern states such as Mississippi, Texas and Alabama, which is on the east part of the core areas of the "tornado alley", increased significantly than first two decades. The Figures (d), (e), (f) show the most recent intensity changes of tornado touchdowns. We can see from the plots that there is a dramatic increase in both the frequency and the intensity of tornadoes on the north part of the "tornado alley" from year 1990 to 2010. It suggests that the frequency of tornado is increasing towards the northern part of "tornado valley" in the state of South Dakota, the boarder of North Dakota and Wisconsin and extends to the Canadian broader. Moreover, the frequency of tornado also increases towards the southeast part of the "tornado alley". In Figures (e) and (f), which is the most recent change in intensity of tornado, southern states such as Florida, Georgia, and Alabama have suffered more incidents of tornado than in 1950s. However, the intensity of tornado show little changes in northern states like New York and Massachusetts, and the east coast.

We can also investigate the total intensity of tornado in the past six decades. The figure below shows the total intensity calculated using kernel smoothing.

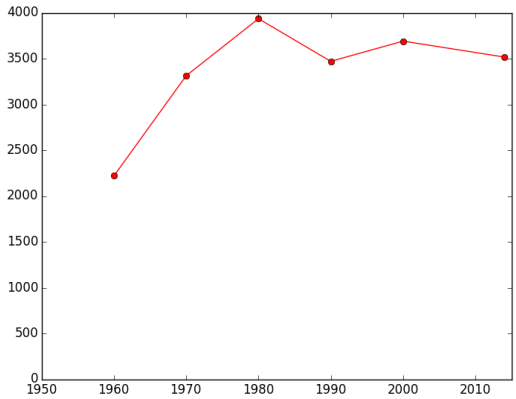


Figure 2: Total Intensity of Tornado Touchdown in the Past Decades

Note that the intensity of tornado starting from 2010 is calculated only based on year 2010-2014. However, we can see that the total intensity is still almost the same as the past years. It is because during 2011 there was a Tornado Super Outbreak, which is the largest tornado outbreak ever recorded. As a result, the total intensity in 2010-2014 still matches the intensity level of the previous decades thanks to the Super Outbreak in 2011.

In conclusion, the result of this project shows that the intensity of tornado shows greater increase in the states that are on the north of the "tornado alley" such as South Dakota and Nebraska. The southern states that are on the east of the "tornado alley" such as Mississippi and Alabama also suffer from an increase in tornado intensity recently. The intensity of tornado shows little change in northern states and the east coast. However, the states in the "tornado alley" like Oklahoma and Texas still suffer high intensity of tornado in the past 50 years, which can be seen in Figure 1. In short, the intensity of tornado touchdowns has a trend to increase towards the north of the "tornado alley" to the Canadian boarder, and towards the east of the "tornado alley" to the southern states of the US.

5 Appendix - Python Code

Please click on the following link for more information
https://github.com/weifhu0124/Undergraduate_Research

References

[1] Tornado History Project
http://www.tornadohistoryproject.com/

[2] Tornado Alley
http://www.si.edu/Content/SE/Educator%20Guides/Tornado_EdGuide_R5.pdf/