JUMP SMOOTHING ALGORITHM

BY MICHAEL TSIAPPOUTAS, PHD

A geospatial territory rating algorithm for smoothing relativities (actuarial factors) at a zip code level is demonstrated with code in R. The inputs are the relativities and exposure at a zip code level; the output is a smoothed relativity at a zip code level. The method is algorithmic, not model-based, which makes it easily implementable, interpretable, and explainable to the Departments of Insurance. It is particularly suitable small to medium sized insurance carriers who have thin, noncredible data at a zip code level. This is because the smoothing is not based on Pure Premium (Loss Cost) or Loss Ratio predictions, which is unreliable with very thin data especially at small coverages, but rather, each zip code relativity is weighted by the exposure-weighted average relativity of all its neighboring zip codes. R code and data available on GitHub.

Introduction

Smoothing territory relativities (factors) at a zip code level is desirable because most Departments of Insurance feel it's a well-established way of eliminating unfair discrimination (no abrupt premium changes for adjacent zip codes); smooth relativities are likely to produce better Pure Premium (ground up) class plan predictions and better Loss Ratio (adjustment) predictions, as indicated by typical decile dual lift charts; are easier for actuarial data scientists to explain to product line partners, because algorithm can be explained in a single slide and maps visually show a gradual convergence of higher relativities towards areas with dense exposure. One problem with all other methods of smoothing is that they are model-based. A model is built and its predictions are used to smooth. But how much would you trust smoothing based on a model built on very noncredible data? The problem is more relevant for small to medium size companies and for optional coverages, for which at a zip code level data is very sparce.

This method is algorithmic, computationally cheap, smoothing happens directly on the actual relativities with no intermediary model step. Because its based on the weighted average of the relativities of the surrounding zip codes, thin data is much less of a problem compared with model-based smoothing methods.

Algorithm

1. Import exposure and relativity by zip code dataset.
2. Import 'zip pairs', a dataset of zip codes and all their neighboring zip codes.
3. Merge (1) and (2).
4. Calculate 'jump', the difference between each zip pair.
5. If jump is small (less than a threshold), replace the neighbor relativity with the main zip relativity. That is to say, ignore any neighbors whose difference with the main zip code is less than a threshold ('jump').
6. Calculate the exposure-weighted average relativity of all zip codes touching the main zip code.
7. Calculate smooth relativity using Equation 1 below.

$$Relativity_{smooth} = Relativity_{main\ zip} \times \left(\frac{Exposure_{zip}}{Exposure_{zip} + Exposure_{avg\ nbr}} \right) + Relativity_{wgt\ nbr} \times \left(\frac{Exposure_{avg\ nbr}}{Exposure_{zip} + Exposure_{avg\ nbr}} \right) \quad \textit{Eq. (1)}$$

Data and Methodology

The data comes from US Census open data (data.gov and census.gov). To demonstrate how flexible this method is, data other than actuarial was used. The relativities are actually age averages by zip code (2010 Census), which were transformed to relativities by dividing out by their average. Exposure balancing (at state level or otherwise) was not performed. It might be necessary depending on how product line uses the territory relativities. The exposure is the population of that zip code divided by the maximum of the entire column.

The first zip code is shown below for demonstration purposes. The zips in columns *nbr* (for neighbor) are all the zip codes adjacent (neighboring or touching) the main code (*zip* column or 90001). The exposure and relativities of the main zip follow, then the exposure and relativities for all the neighbors. The *jump* threshold was set at 0.01, so the only *nbr* value that was replaced with the main *zip* value is the second row (yellow cells). When we calculate the weighted averages, we have only one row left (below). We see that the main zip code relativity used to be 0.7146 and the resulting smooth version of it (after the first iteration) is 0.7186. This is because the average of the surrounding zip relativities are 'pulling' the main relativity higher (average with the higher relativity of 0.7224).

First zip code with all its neighbors (pairs1 dataset)

zip	nbr	zip.exp	zip.rel	nbr.exp	nbr.rel	jump	nbr.rel.jump
90001	90002	0.5411	0.7146	0.4853	0.6850	0	0.6850
90001	90003	0.5411	0.7146	0.6278	0.7065	1	0.7146
90001	90011	0.5411	0.7146	0.9843	0.7038	0	0.7038
90001	90058	0.5411	0.7146	0.0305	0.6984	0	0.6984
90001	90255	0.5411	0.7146	0.7112	0.7817	0	0.7817

First zip code after calculating smooth relativity (wtd.avgs1 data set)

zip	zip.rel	zip.exp	wtd.nbr.rel	avg.nbr.exp	smooth.rel1
90001	0.7146	0.5411	0.7224	0.5678	0.7186

In a nutshell, the smooth relativity is the average between the main zip code relativity and the exposure-weighted average of those neighboring zip code relativities whose absolute difference with the main zip code is above a threshold (jump).

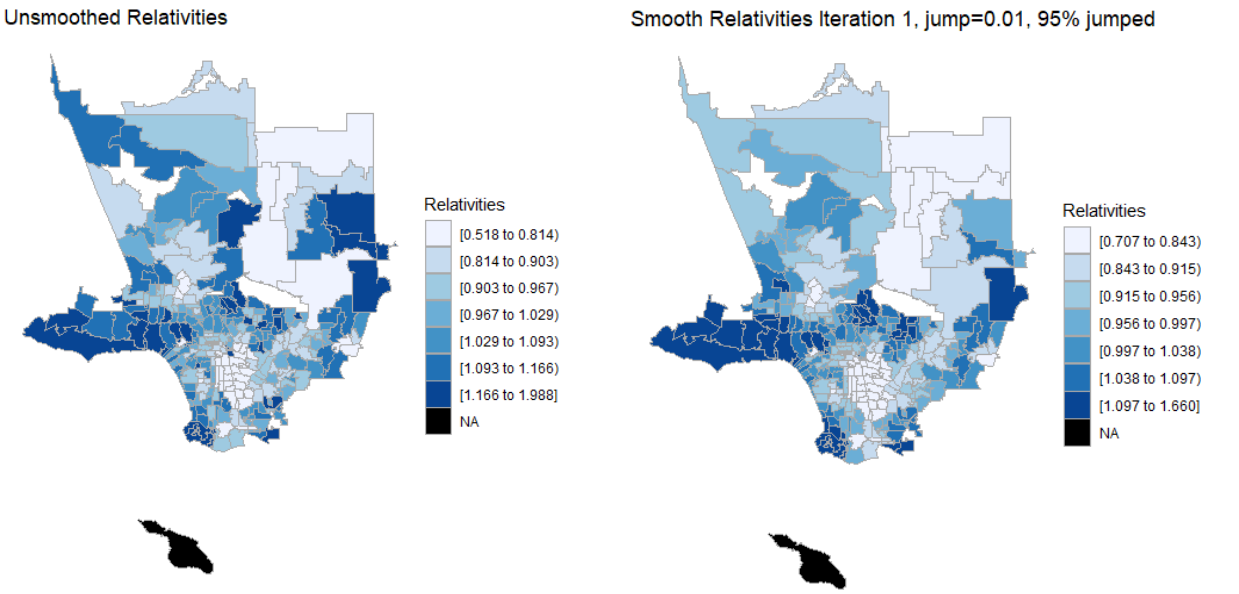

Figure 1. *Downtown Los Angeles CA. Unsmoothed above and Smoothed Age Relativities below. The black (NA) area is zip code 90704, as indicated by the warning* "The following regions were missing and are being set to NA: 90704".

Why the "Jump"?

The jump is a way to control how much smoothing happens within each iteration. Think of it as the "shrinkage" in tree algorithms. The higher the jump coefficient, the less smoothing happens, because more neighboring zip codes are ignored, or replaced by the main zip code's relativity. So that the resulting exposure-weighted average of the surrounding zip codes is closer to the main zip code relativity, and therefore the neighbors don't 'pull' the main as much anymore.

For example, Figure 2 shows jump coefficients of 0.01, 0.1, and 0.4, which yielded jump rates (how many cases had a difference more than the threshold) of 95%, 47%, and 6% respectively. The higher the threshold, the less the jump rate, which means fewer zip codes where used in the actual averaging, which yields less smoothing.

We suggest choosing a jump threshold that yields about 50% jump rate and proceed with the next iteration. That way smoothing happens at a comfortable pace (neither too fast or too slow) and iterations have a chance to converge nicely.

Iterations

Iterations are simply the smoothed version of the already smoothed relativities. The first iteration is the first smoothing. The second iteration is the smoothing of smoothed relativities from the first iteration.

Figure 3 shows how the first three iterations look like. For all iterations, the jump threshold was tweaked to yield a rate of about 50%.

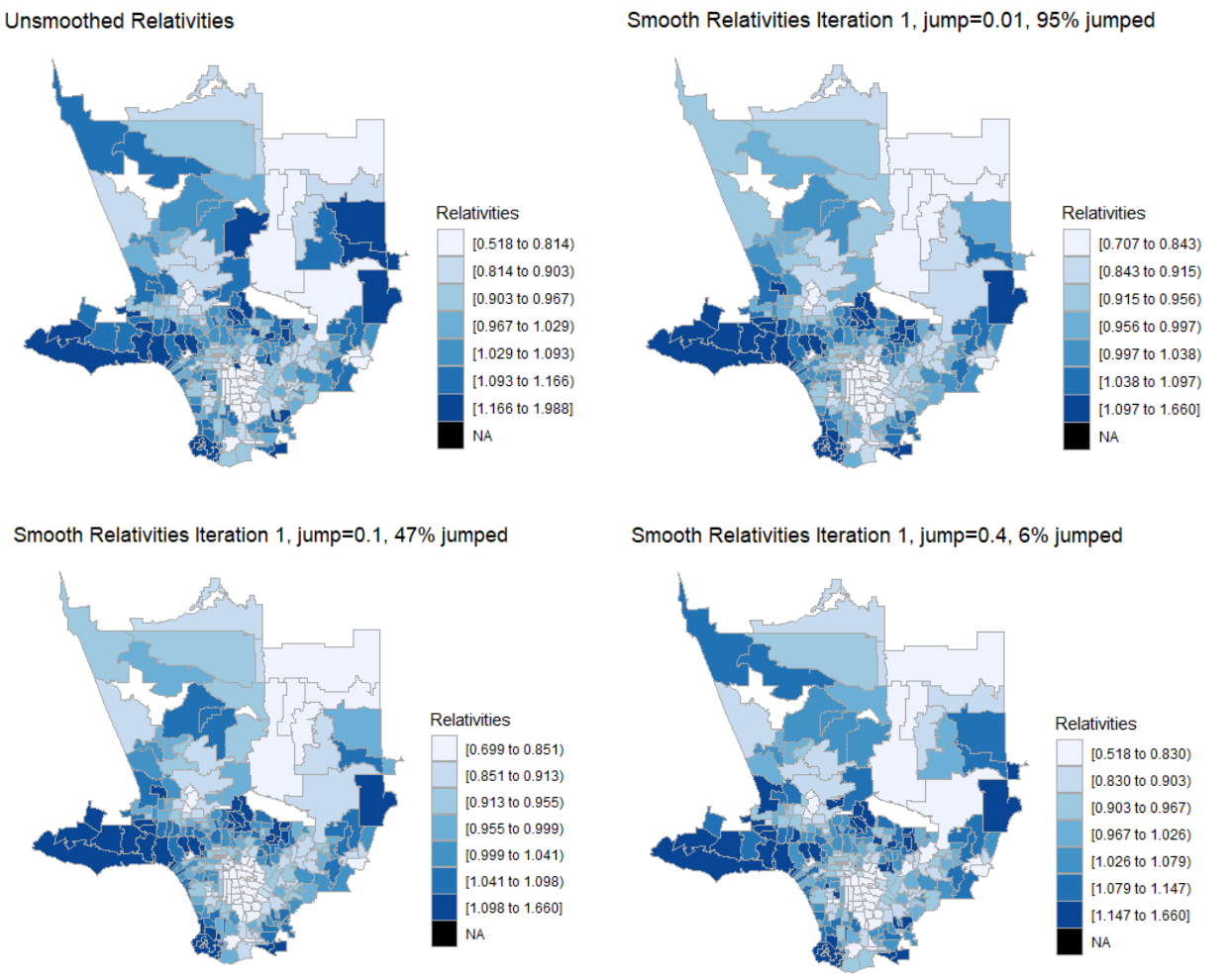

Figure 2. *How jump affects smoothing within one (here first) iteration. The higher the jump threshold, the less smoothing happens, because less neighboring zip codes are taken into account.*

The jump rate of about 50% is by no means a rule, not even a rule of thumb. It's just a way to control how much smoothing happens with each iteration. For some applications, slow smoothing within each iteration and more iterations works better than less iterations with each smoothed aggressively. The proverbial 'part science, part art' is true here!

Even though there's a good dose of art to this method, we suggest using at least a handful of metrics established within your organization to help you decide a good balance of iterations vs jump. Visualization is a good place to start, however, because it gives you a quick sense of how fast your data converge, i.e., how many iterations it takes to reach an asymptote of no significant smoothing happening anymore.

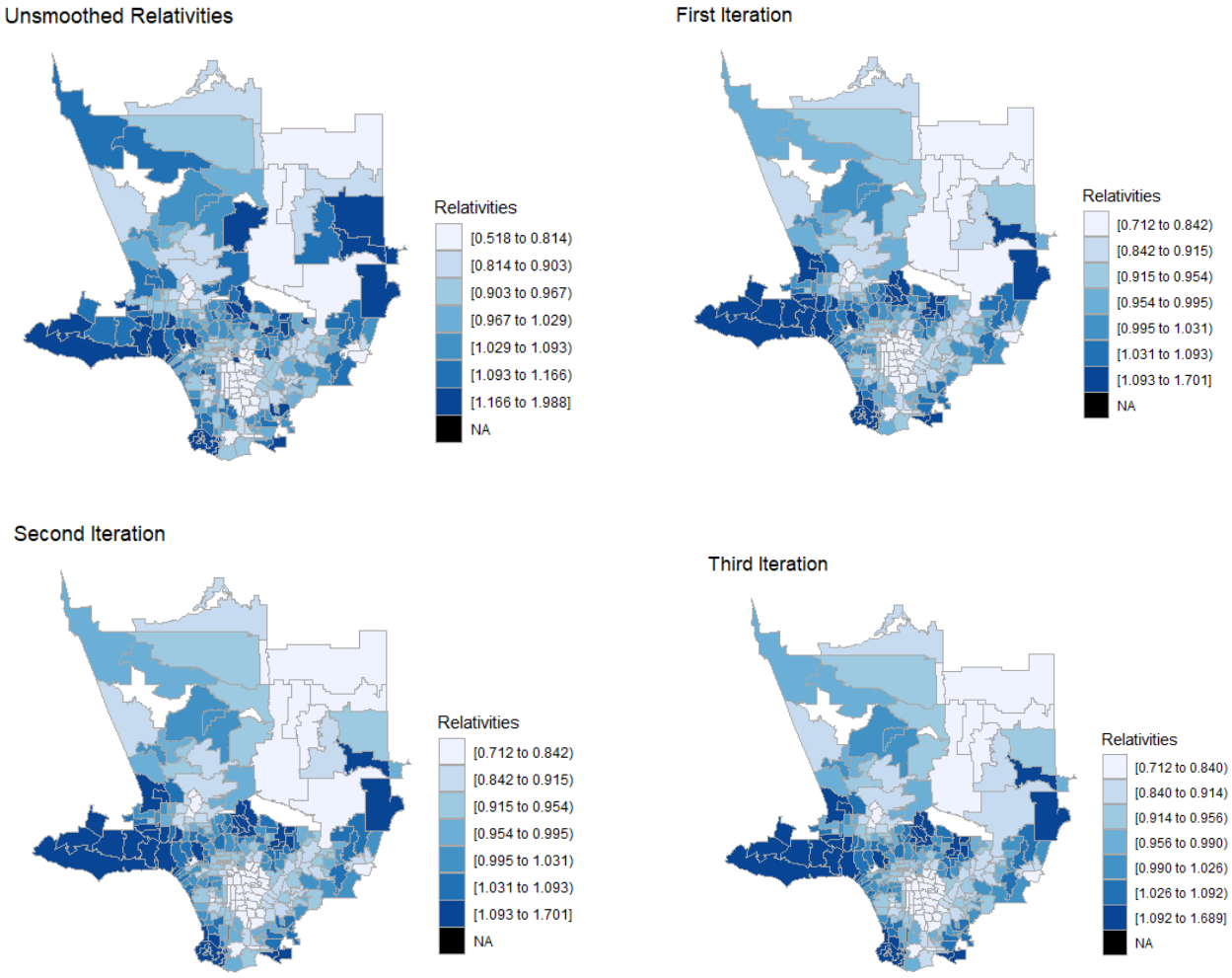

Figure 3. *The first three iterations, each with jump rate of about 50%.*

How Many Iterations are Enough?

A very heuristic measure of when to stop is when your jump threshold coefficient reaches an asymptote. Since the jump rate is kept approximately constant at 50%, with each successive iteration, the difference between the main and smooth relativities required to achieve this 50% jump rate, should shrink, because with more iterations, the relativities of main zips and their neighbors come closer and closer to each other. An example of a jump coefficient plot is shown in Figure 4. A quick look justifies our decision to stop at the third iteration.

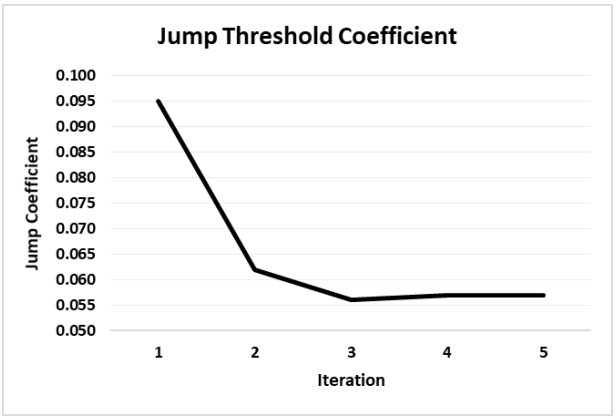

Figure 4. *Jump Threshold Coefficients*

Code and data

The R code used to generate all results and figures in this paper, along with its data, and this white paper are all available on GitHub under the GNU 3.0 Public License.
https://github.com/tsiappoutas/smoothing

General Info

```
##############################################################################
# white paper, this code, and data available at                              #
# https://github.com/tsiappoutas/smoothing                                   #
# author: michael tsiappoutas                                                #
# email: tsiappoutas@hotmail.com; tsiappoutas@gmail.com                      #
# last update: Oct. 2020                                                     #
##############################################################################
```