StatLearn / Homework 01

Pierpaolo Brutti

Due Thursday, April 12, 2018, 23:00 PM on Moodle

General Instructions

I expect you to upload your solutions on Moodle as a **single running** `R Markdown` file (`.rmd`), named with your surnames. In case you decide to work in Python or other programming language, you still have to provide a **running**, **sharable** notebook with detailed instructions on how to use it, **plus** a doc with the final result.

R Markdown Test

To be sure that everything is working fine, start `RStudio` and create an empty project called `HW1`. Now open a new `R Markdown` file (`File > New File > R Markdown...`); set the output to `HTML mode`, press `OK` and then click on `Knit HTML`. This should produce a web page with the knitting procedure executing the default code blocks. You can now start editing this file to produce your homework submission.

Please Notice

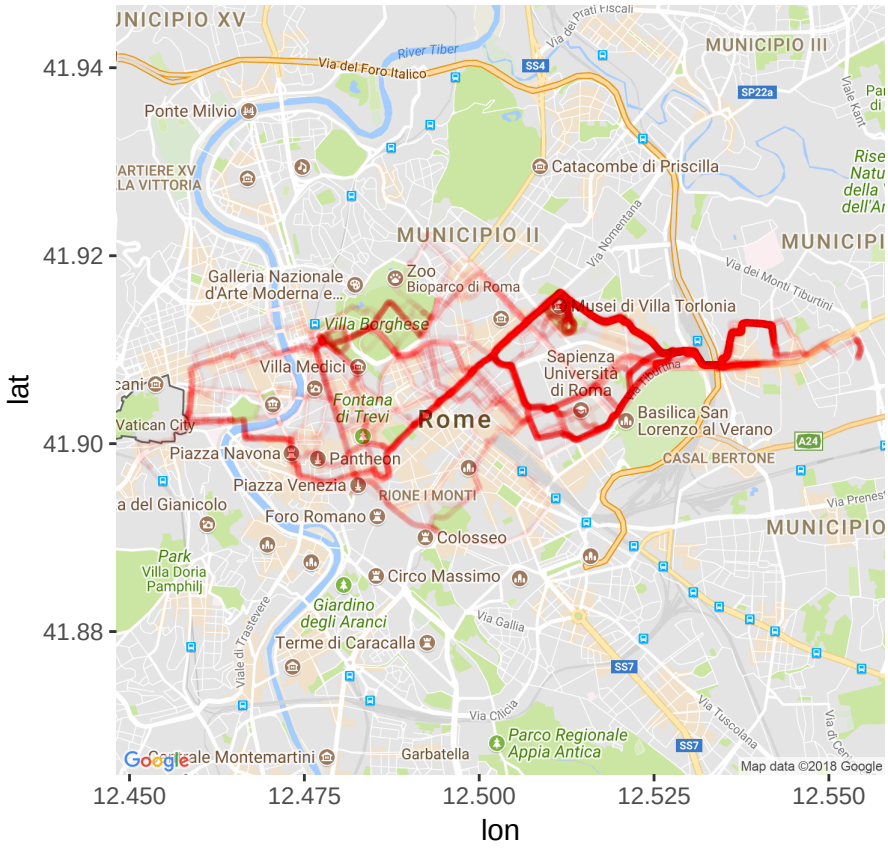
- For more info on `R Markdown`, check the support webpage that explains the main steps and ingredients: R Markdown from RStudio.
- For more info on how to write math formulas in LaTex: Wikibooks.
- Remember our **policy on collaboration**: *Collaboration on homework assignments with fellow students is **encouraged**. However, such collaboration should be clearly acknowledged, by listing the names of the students with whom you have had discussions concerning your solution. You may **not**, however, share written work or code after discussing a problem with others. The solutions should be written by **you**.*

Track me if you can...

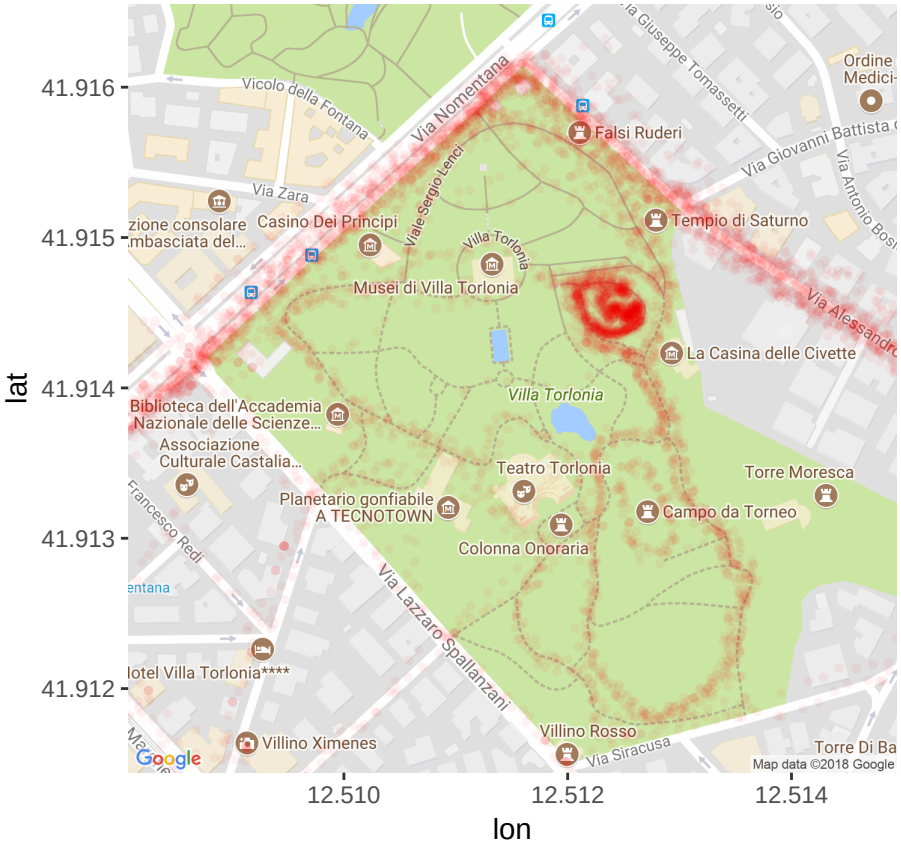
Introduction

Personal data coming from (personal) tracking devices are massive these days. So, let's take a very... *personal*... point of view on the issue. The dataset `trackme.RData` collects the gps info of 60 of my running sessions. To take a look we can use the `ggmap` package as follow:

```
# The package
require(ggmap)

# The data
load("trackme.RData")

# Map boundaries
myLocation <- c(min(runtrack$lon, na.rm = T),
                min(runtrack$lat, na.rm = T),
                max(runtrack$lon, na.rm = T),
                max(runtrack$lat, na.rm = T))

# Get the map from Google (default)
myMapInD <- get_map(location = myLocation, maptype = "roadmap", zoom = 13)

# Plot gps coordinates (without elevation data)
gp <- ggmap(myMapInD) + geom_point(data = runtrack,
                                   aes(x = lon, y = lat),
                                   size = .5, colour = I("red"), alpha = .01)
# Take a look
print(gp)
```




```
# Zoom in / Restricted map boundaries
reLocation <- c(12.508, 41.911, 12.515, 41.917)
# Get the map from Google (default) and plot
reMapInD <- get_map(location = reLocation, maptype = "roadmap")
ggmap(reMapInD) + geom_point(data = runtrack, aes(x = lon, y = lat),
                             size = 1, colour = I("red"), alpha = .05)
```



Your job / Part I

1. First of all, considering only the lon–lat information, treat `runtrack` simply as a 2D point cloud in the Euclidean space and use **any** method, `R` package and function you like to **estimate** and **visualize** properly and meaningfully the density of this data (... **boldface** is there for a reason). Please notice: this dataset is medium sized but you can still have problems of speed, be smart and do your best to get around them.

2. Can you figure out a way to single out the places where I run the most? One option is the **mean-shift**, can you do better? Conclude **commenting** your results. For the sake of completness, let us briefly review the *mean-shift* here. So, given data $\{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n\} \overset{\text{IID}}{\sim} p$, we construct an estimate $\widehat{p}(\cdot)$ of the density. Let $\{\widehat{\boldsymbol{m}}_1, \ldots, \widehat{\boldsymbol{m}}_K\}$ be the estimated modes, and let $\{\widehat{\mathcal{A}}_k\}_{k=1}^K$ be the corresponding ascending manifolds implied by $\widehat{p}(\cdot)$. The sample or empirical clusters $\{C_1, \ldots, C_K\}$ are then defined as

$$C_k = \{\boldsymbol{X}_i \,:\, \boldsymbol{X}_i \in \widehat{\mathcal{A}}_k\}, \quad \forall\, k \in \{1, \ldots, K\}.$$

To be more specific, consider a (isotropic) kernel density estimator

$$\widehat{p}_h(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{h^d} \mathsf{K}\left(\frac{\|\boldsymbol{x} - \boldsymbol{X}_i\|}{h}\right).$$

To locate the mode of $\widehat{p}_h(\boldsymbol{x})$ we use the *mean-shift algorithm* which finds modes by approximating the steepest ascent paths. The algorithm consists of the following steps:

Input: $\widehat{p}_h(\boldsymbol{x})$ and a mesh of points $A = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N\}$ often taken to be the data points.

Inits: For each point $\boldsymbol{a}_j$, set $\boldsymbol{a}_j^{(0)} = \boldsymbol{a}_j$.

Iterate: Until convergence the following equation

$$\boldsymbol{a}_j^{(s+1)} \leftarrow \sum_{i=1}^{n} \left[\frac{\mathsf{K}\left(\frac{\|\boldsymbol{a}_j^{(s)} - \boldsymbol{X}_i\|}{h}\right)}{\sum_{r=1}^{n} \mathsf{K}\left(\frac{\|\boldsymbol{a}_j^{(s)} - \boldsymbol{X}_r\|}{h}\right)}\right] \boldsymbol{X}_i.$$

Output: $\widehat{\mathcal{M}} = \texttt{unique}\{\boldsymbol{a}_1^{(\infty)}, \ldots, \boldsymbol{a}_N^{(\infty)}\}$.

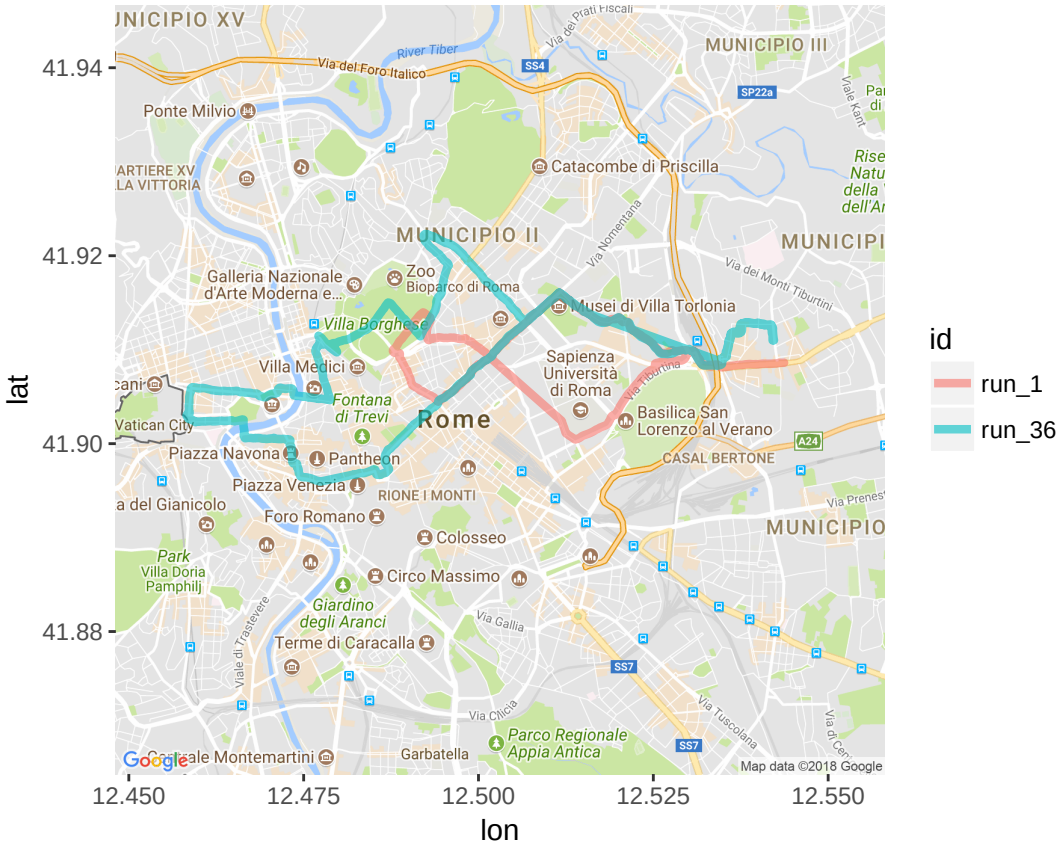
The result of this algorithm is then the set of estimated modes $\widehat{\mathcal{M}}$ together with the actual clustering: the mean-shift algorithm, in fact, shows us exactly what mode each point is attracted to![1]

Let's be honest...

Those red dots are not really dots, they are part of a single track, a single curve in the plane. Let's look at two of them:

```
# Plot gps coordinates (without elevation data)
runsmall <- subset(runtrack, id %in% c("run_1", "run_36"))
gp2 <- ggmap(myMapInD) + geom_path(data = runsmall,
                                   aes(x = lon, y = lat, col = id),
                                   size = 1.5, lineend = "round",
                                   alpha = .6)
# Take a look
print(gp2)
```

[1] More formally, the mean-shift is simply tracing the so called *gradient flow*. The flow lines lead to the modes and define the clusters. In general, a *flow* is a map $\phi : \mathbb{R}^d \times \mathbb{R} \mapsto \mathbb{R}^d$, where the second argument can be interpreted as time, such that: 1. $\phi(\boldsymbol{x}, 0) = \boldsymbol{x}$; 2. $\phi\big(\phi(\mathbf{x}, t), s\big) = \phi(\mathbf{x}, s + t)$. The latter is called the *semi–group property*.



We are in the realm of what is now called **functional data analysis**, a broad name to denote a branch of statistics that starts from the very idea that a single datapoint, a single observation is **not** just a number or a label as we use to think, but a more complex beast, *usually* a whole function that we observe sampled at a given frequency. A single observation X then, is *not* just high dimensional, it is **infinite** dimensional: in our case each X_i is a *curve*. An immediate problem is that the very concept of a density $p(\cdot)$ over such a space is no longer well defined but *geometry* can help us. As we did in our notes, geometrically, we can think of $p(\cdot)$ as

$$p(x) = \lim_{\varepsilon \to 0} \frac{\Pr(\|X - x\| \leqslant \varepsilon)}{\texttt{volume of a ball with radius } \varepsilon}.$$

Now, when the outcome space $\mathcal{X}$ of our experiment is a set of curves, we may still define the density geometrically by

$$q_\epsilon(x) = \Pr\big(\text{dist}(x, X) \leqslant \epsilon\big),$$

where $\text{dist}(\cdot, \cdot)$ is some metric, some distance on $\mathcal{X}$; that is, between curves. However we cannot divide by the volume of the ε-sphere in $\mathcal{X}$ and let ϵ tend to zero since the dimension d of $\mathcal{X}$ is *infinite*! One simple way around this is to use a fixed ε and work with the **unnormalized** density $q_\epsilon(\cdot)$. For the purpose of finding high–density regions (...and then clustering, for example...) this may be adequate. An estimate of $q_\epsilon(\cdot)$ is

$$\widehat{q}_\epsilon(x) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}\big(\text{dist}(x, X) \leqslant \epsilon\big),$$

but in fact, we can substitute the indicator function with any other smoothing kernel we know!

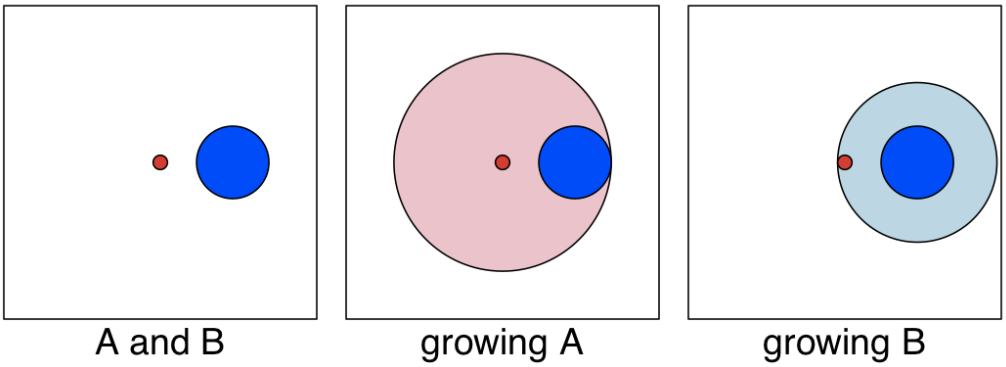
OK, now back to our exercise. The starting point is: *each data point is a curve in* $\mathbb{R}^2$. Various questions are of interest: *Where are the tracks most common? Are there clusters of tracks? Is the density of tracks changing over time?*

Each curve X_i is identified by the level of the variable `id` in the dataframe `runtrack` and can be regarded as a mapping $\boldsymbol{X}_i : [0, T_i] \mapsto \mathbb{R}^2$ where $\boldsymbol{X}_i(t) = \big(X_{i_1}(t), X_{i_2}(t)\big)$ is my position at time t and T_i is the duration of the i^{th} run. Now, for any $i \in \{1, \ldots, n = 60\}$, let's associate to run i, the **set**

$$X_i \quad \rightsquigarrow \quad \mathsf{G}_i = \Big\{ \boldsymbol{X}_i(t) = \big(X_{i_1}(t), X_{i_2}(t)\big) \text{ such that } t \in [0, T_i] \Big\}, \tag{1}$$

that is, the graph of curve X_i. Once we look at our *curve* as a *set*, we con use the Hausdorff metric to measure distances (but feel free to move to another suitable metric if you like). Denoting the *enlargement* of the set A by $A^\varepsilon = \bigcup_{x \in A} B(x, \varepsilon)$ with $B(x, \varepsilon)$ being an ε–ball centered at x as usual, in class we defined the Hausdorff distance between two sets A is B as

$$\text{dist}_H(A,B) = \{\text{smallest value } \varepsilon > 0 \text{ such that } A \subset B^{\varepsilon} \text{ and } B \subset A^{\varepsilon}\} = \max\left\{\sup_{x\in A}\inf_{y\in B}\text{dist}(x,y),\ \sup_{x\in B}\inf_{y\in A}\text{dist}(x,y)\right\} \quad (2)$$



Then, for a fixed $\epsilon > 0$ and any planar curve γ, our *unnormalized* boxcar-kernel estimator is

$$\widehat{q}_{\varepsilon}(\gamma) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{I}\big(\text{dist}_H(\gamma, \mathsf{G}_i) \leqslant \varepsilon\big). \quad (3)$$

Equation (3) together with Equation (2) seem obscure at first, but let us try to focus on the very nature of their basic ingredients once specialized to our setup, shall we?

1. First of all, although γ can be any planar curve, let's take it equal to one of the data-point, say $\gamma = \mathsf{G}_j$.
2. Now we should calculate $\text{dist}_H(\mathsf{G}_j, \mathsf{G}_i)$.
3. Notice that $\text{dist}_H(\mathsf{G}_j, \mathsf{G}_i)$ depends only on $\text{dist}(x,y)$, the distance between *single elements* in G_j and G_i.
4. What are these "single elements" here? From Equation (1) we know they are $x \leftarrow \big(X_{i_1}(t), X_{i_2}(t)\big)$ and $y \leftarrow \big(X_{j_1}(t), X_{j_2}(t)\big)$ for a fixed t; that is, points in $\mathbb{R}^2$...
5. ...and we know how to evaluate distances between dots in the plane: *Euclidean distance*![2]

...after this, it is just a question of taking min's and a bunch of max's. In other words, the *in-sample* evaluation of $\widehat{q}_{\varepsilon}(\cdot)$ is maybe computationally heavy, but relatively easy. On the other hand, the *out-of-sample* evaluation is a bit trickier but, in the end, it just amounts at *sampling* the curve γ we target at specific locations/time-points and, if you think about it, this is actually what we do *all the times* once we try to plot it in `R` or any other software. So, in the end, whatever the planar curve γ is: **if you can plot it, you can estimate at it!**

One final comment: of course we have to choose ε and of course we can explore a bit by tracking what happen for different values on a grid $\mathcal{E} = \{\varepsilon_1, \ldots, \varepsilon_m\}$. On the other end we can also try to anchor ε to some tangible quantity like the *quantiles* of the set of distances

$$\mathcal{D}_n = \Big\{\text{dist}_H(\mathsf{G}_i, \mathsf{G}_j),\ \text{for } i \neq j\Big\}.$$

Your job / Part II

Essentially: "*implement the previous idea at your best in any language you like, and explore the data both numerically and visually*". For example:

1. Pick a possibly meaningful ε (and/or play around with different ε's). With this choice...
2. ...find the top-5 paths with the highest local density,
3. ...find the bottom-5 paths with the lowest local density,
4. ...now that you have a kernel estimator, you can run **mean-shift**[3] and cluster my tracks...

In the end, whatever you do: comment, comment, comment, comment, comment!

[2] In our case, although we are dealing with *latitute-longitude* pairs, we can still rely on a flat geometry...I'm not runnig across the globe really!
[3] There is also a paper available. You don't need to read it, but if you are curious...