Università Cattolica Sacro Cuore

Statistical and Actuarial Sciences

mj: Data Business Analytics

Spatial Machine Learning modelling: End-to-End web App solution

Author:
Niccolò Salvini

Supervisor:
Dr. Marco DellaVedova

Assistant Supervisor:
Dr. Vincenzo Nardelli

AY 2019 / 2020

End-to-End Real Estate Rental app, a Bayesian spatial modelling approach wtih INLA

Niccolò Salvini[1]

date: Last compiled on 26 settembre, 2020

[1]https://niccolosalvini.netlify.app/

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Main themes:

- General introduction to the problem to solve
- Open Data discussion
- Research Question
- Milan Real Estate Controversies
-
- Why a Bayesian approach

Chapter 2

Scraping

2.1 What is Web Scraping

Web scraping is a techniques aimed to extract data from static or dynaminc internet web pages. It can be done automatically and simultaneously. Given the substantial unavailability of fresh data regarding Milan Real Estante rental markets, this practices have to be forcibly applied. Data in website appears to be well organized and easily accesible, nevertheless sometimes they are not Lo web scraping è una tecnica di estrazione dei dati da pagine internet statiche o dinamiche in maniera automatica e simulatanea (Wikipedia, 2020). L'impossibilità di reperire dati aperti aggiornati riguardo l'affitto sul mercato italiano mi ha spinto a sviluppare sofisticate tecniche di estrazione di dati orientate ad alleggerire lo sforzo e aumentare la velocità di reperimento: da una parte nel preprocessing del dataset, nella successiva del fragente del modelling, per finire con la reattività di risposta dell'applicazione. Le informazioni sui siti appaiono spesso ordinate e semplici, tuttavia ogni sito web ha una propria architettura e un proprio linguaggio. Per architettura intendo struttura gerarchica secondo cui è organizatto un sito internet: una semplificazione della struttura di un sito web può essere un insieme di cartelle innestate una dentro l'altra collegate tra loro da riferimenti tramite l'url. la natura gerarchica della struttura prevede che si usi un linguaggio che fa propria questa caratteristica,

HTML è il preferito. L'html si organizza in nodi ed angoli, esattamente come un grafo; che aggiunta la componente gerarchica fa sì che questo sia un albero. Difatti spesso ci si riferisce alla struttura delle pagine web come html tree. Ogni elemento nella pagina ha un suo preciso posto nel codice sorgente della stessa e ha un preciso valore o più valori. Possiamo immaginare ogni nodo della pagina come una lista di valori che è collegata ad un nodo precedente detto padre da una struttura gerarchica superiore, ed eventiualmente ad un nodo successivo detto figlio. Pertanto tutte le informazioni che giacciono sotto al nodo padre sono parenti del nodo padre e sono direttamente collegate (directed nel senso dell'interpretazione), parallelamente ci saranno altri nodi padre che saranno adiacenti al nodo padre, i quali avranno nodi figli e così via. La complessità della pagina e del codice è tanto maggiore quanto il livello dell'albero aumenta, tanto più l'albero è folto tanto più sarà difficile individurare il ramo o la foglia che ci interessa. Ragionevolemnte accade lo stesso per la funzione di scraping e il tempo di scraping. Html organizza i contenuti e le relazioni tra loro, il css (Cascading Style Sheets) invece si occupa dello stile e della formattazione degli stessi. il css è uno strumento molto potente in mano ad uno scraper perchè permette di recuperare informazioni simili tra loro ma che occupano nodi con posizione gerarchica diversa all'interno della pagina. Pertanto una volta letto l'html della pagina sarà necessario recuperare la query css per raccogliere tutti gli elementi di interesse tramite la funzione di scraping. Successivamente occorre notare che l'encoding da html a stringa di testo non è quasi mai lineare, spesso occorre riformattare, cancellare spazi, convertire la natura dell'oggetto estratto etc. Il successivo elemento di complessità incontrato durante questa prima fase è stato interfacciarsi con un server attento alle richieste GET degli utenti. I dati viaggiano in pacchetti da un server che ospita un sito internet al nostro laptop. tutte le volte che cerchiamo di accedere ad un sito stiamo mandando una richiesta di ricezione di pacchetti dati ad un server in qualche luogo remoto del mondo. Quandi bussiamo alla porta del server se non siamo sospetti e superiamo i criteri autostabiliti dal server questo risponde, e lo fa con un numero che spazia da 200 a 500, due esempi: 200 se la risposta è positiva,

404 se la risposta è negativa. I criteri secondo cui gli utenti sono calssificati secondo utente normale o utente sospetto (aka bot) sono sintetizzati in un documento di testo chiamato robot.txt. Questo file di testo raccoglie tra le altre due infromazioni principali il delay time, cioè il tempo preferito dal server che deve intercorrere tra una richiesta dati e la successiva e quale utente è autorizzato ad accedere. Ogni utente posside un indirizzo IP che nelle richieste a server si codifica in user agent, cioè una stringa di testo dove vengono raccolte le infromzioni significative circa il dispositivo da cui provengono le richieste, un esempio:

'Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36',

dove ogni segmento della stringa rispecchia una caratteristica del laptop del richiedente, Chrome/54.0.2840.71 è la versione del browser chrome da cui proviene la richiesta Safari/537.36', è il motore di ricerca etc.

2.2 Scraping Best Practices and Robot.txt

Robots.txt files are (rivedi citation) a way to kindly ask webbots, spiders, crawlers, wanderers and the like to access or not access certain parts of a webpage. The de facto 'standard' never made it beyond a informal "Network Working Group INTERNET DRAFT". Nonetheless, the use of robots.txt files is widespread (e.g. https://en.wikipedia.org/robots.txt, https://www.google.com/robots.txt) and bots from Google, Yahoo and the like will adhere to the rules defined in robots.txt files, although their *interpretation* of those rules might differ.

Robots.txt files are plain text and always found at the root of a website's domain. The syntax of the files in essence follows a fieldname: value scheme with optional preceding user-agent: … lines to indicate the scope of the following rule block. Blocks are separated by blank lines and the omission of a user-agent field (which directly corresponds to the HTTP user-agent field) is

seen as referring to all bots. # serves to comment lines and parts of lines. Everything after # until the end of line is regarded a comment. Possible field names are: user-agent, disallow, allow, crawl-delay, sitemap, and host. For further notions (Meissner and Ren, 2020, goo (2020))

Some interpretation problems:

- finding no robots.txt file at the server (e.g. HTTP status code 404) implies that everything is allowed
- subdomains should have there own robots.txt file if not it is assumed that everything is allowed
- redirects involving protocol changes - e.g. upgrading from http to https - are followed and considered no domain or subdomain change - so whatever is found at the end of the redirect is considered to be the - robots.txt file for the original domain
- redirects from subdomain www to the doamin is considered no domain change - so whatever is found at the end of the redirect is considered to be the robots.txt file for the subdomain originally requested

For the thesis purposes it has been designed a dedicated function to inspect whether the domain requires specific actions or prevents some activity on thw target website. The following `checkpermission()` function has been integrated inside the scraping architecture and it is called once at the very beginning.

```
library(robotstxt)
dominio = "immobiliare.it"

checkpermission = function(dom) {

    robot = robotstxt(domain = dom)
    vd = robot$check()[1]
    if (vd) {
```

```
        cat("\nrobot.txt for", dom, "is okay with scraping!")
    } else {
        cat("\nrobot.txt does not like what you're doing")
        ## stop()
    }
}
checkpermission(dominio)
```

```
##
## robot.txt for immobiliare.it is okay with scraping!
```

Further improvements in this direction came from the `polite` package (Perepolkin, 2019) which combines the power of the `robotstxt`, the `ratelimitr` to rate-limiting requests and the `memoise` for response caching. This package is wrapped up around 3 simple but effective ideas:

> The three pillars of a polite session are seeking permission, taking slowly and never asking twice.

The three pillars constitute the Ethical web scraping manifesto (Densmore, 2019) which are common shared practises that are aimed to self regularize scrapers. This has not nothing to do with law but since many scrapers themselves, as website administrators or analyst, have fought with bots. Bots might fake out real client logs and might stain analytics, so here it is born the choice to fine common ground and politely ask for permission.

2.3 User agents, Proxies, Handlers

Everytime a user enters a website what he is really doing is sending an HTTP request to the website server with some information packed. This can be easily thought as a generic person A that rings the door's bell of person B's house. A

comes to the B door with its personal information, its name, surname, where he lives etc. At this point B may either answer to A requests by opening the door and let him enter given the set of information he has, or it may not since B is not sure of the real intentions of A. This typical everyday situation in nothing more what happens billions of times on the internet everyday, the user (in the example above A) is interacting with a server website (part B) sending packets of information. If a server does not trust the information provided by the user, if the requests are too many, if the requests seems to be scheduled due to fixed sleeping time, a server can block the requests. In certain cases it can even forbid the user to be on the website. The language the two parties talks are coded in numbers that ranges from 100 to 511, each of which ha its own significance. A popular case of this type of interaction occurs when users are not connected to internet so the server responds 404, page not found. Servers are built with a immune-system like software that raises barriers and block users to prevent dossing or other illegal practices.

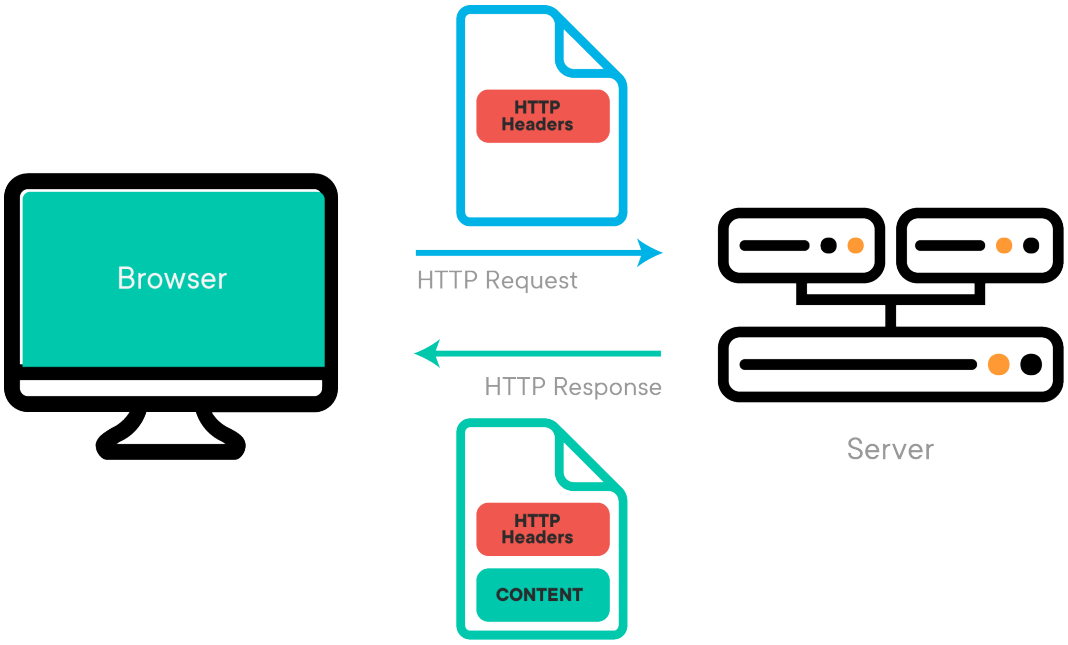


Figure 2.1: How Web Works

This procedure is a daily issue to people that are trying to collect information from websites. Google does it everyday with its spider crawlers, which are very sophisticated bots that performs scarping over a enormous range of

websites. This challenge can be addressed in multiple ways, there are some specific Python packages that overcome this issue. The are also certain types of scraping as the Selenium web driver automation that simulates browser automation. Selenium allows the user not to be easily detected by the server immune system and peaceful. In here precautions have not been taken lightly, and a simple but effective approach is proposed.

2.3.1 User agents Spoofing

A user agent (who, 2020) is a string of characters in each browser that serves as an identification agent. The user agent permits the web server to be able to identify the user operating system and the browser. Then, the web server uses the exchanged information to determine what content should be presented to particular operating systems and web browsers on a series of devices. The user agent string contains the user application or software, the operating system (and their versions), the web client, the web client's version, and the web engine responsible for the content display (such as AppleWebKit). The user agent string is sent in form of a HTTP request header. Since the User Agents acts as middle man between the client request and the server response what it would be better doing is to actively faking it so that each time a web browser presents himself to a web server it has a different specifications, different web client, different operating system and so on.

The simple approach followed was building a vector of samples of different existing and updated User Agents (UA). Then whenever a request from a browser is served to a web server 1 random string is drawn from the user agents pool. So each time the user is sending the requests it appears to have a different "identity". Below the user agents rotating pool:

```
set.seed(27)
agents = c("Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.116 Safari/537.36",
    "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.116 Safari/537.36",
    "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.116 Safari/537.36",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/602.2.14 (KHTML, like Gecko) Version/10.0.1 Safari/602.2.14",
    "Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36",
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.98 Safari/537.36",
```

```
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_6) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.98 Safari/537.36",
    "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.71 Safari/537.36",
    "Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/54.0.2840.99 Safari/537.36",
    "Mozilla/5.0 (Windows NT 10.0; WOW64; rv:50.0) Gecko/20100101 Firefox/50.0")
agents[sample(1)]
```

```
## [1] "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/83.0.4103.116 Safari/537.36"
```

An improvement to this could be using also rotating proxies. A proxy server acts as a gateway between the user and the server. It's an intermediary server himself, separating end clients from the websites they are browsing. Proxy servers provide varying layers of functionality, security, and privacy are some of the examples. While the user is exploiting a proxy server, internet traffic flows through the proxy server on its way to the server you requested. The request then comes back through that same proxy server and then the proxy server forwards the data received from the website to you. Many proxy servers are offered in a paid version so in this case since security barriers of the target website are not high they will not be implemented. It has to be mentioned that many online services are providing free proxies but the turnaround of this solutions are many, two of them are: - Proxies to be free are widely shared among people, so as long as someone has used them for illegal purposes the user is inheriting their mistakes when caught. - Some of those proxies, pretty all the ones coming from low ranked websites, are tracked so there might be a user privacy violation issue.

2.3.2 Handlers

During the scraping many things could be going wrong. Starting from the things that have been previously explained at the chapter start (URL structure changes, data are moved in different location so that css query goes empty…), ending with the ones that have just been said a few lines ago. Handlers and trycatch error workarounds are explicitly built in this sense. The continuous testing of the scraping functioning while developing has required the maintainer to track where the error occurs. A few numbers: the "agglomerative"

function `get.data.catsing()` triggers more than 36 scrapping functions that are going to catch 36 different data pieces. If one of them went missing then the other one would be missing too. Then when row-data is binded together one entry column might not exists making the process fail.
Then the solution to that is to call inside the aggolmerative function as much as trycatch as many scrapping functions are involved. The trycatch can leverage the gap by introducing a specified quantity and alerting that something went wrong. On top of that many other handlers are called throughout the procedure:

- `get_ua()` verifies that the user agent coming from the session request is not the default one

```
get_ua = function(sess) {
    stopifnot(is.session(sess))
    stopifnot(is_url(sess$url))
    ua = sess$response$request$options$useragent
    return(ua)
}
```

- `is_url()` verifies that the url input needed has the canonic form. This is done by a REGEX query.

```
is_url = function(url) { re =
    "^(?:(?:http(?:s)?|ftp)://)(?:\\S+(?::(?:\\S)*)?@)?(?:(?:[a-z0-9¡-<ef><U+00BF><U+00BF>](?:-)*)*(?:[a-z0-9¡-<ef><U+00BF><U+00BF>])+)(?:\\.(?:[a-z0-9¡
    grepl(re, url) }
```

- `.get_delay()` checks through the robotxt file if a delay between each request is kindly welcomed.

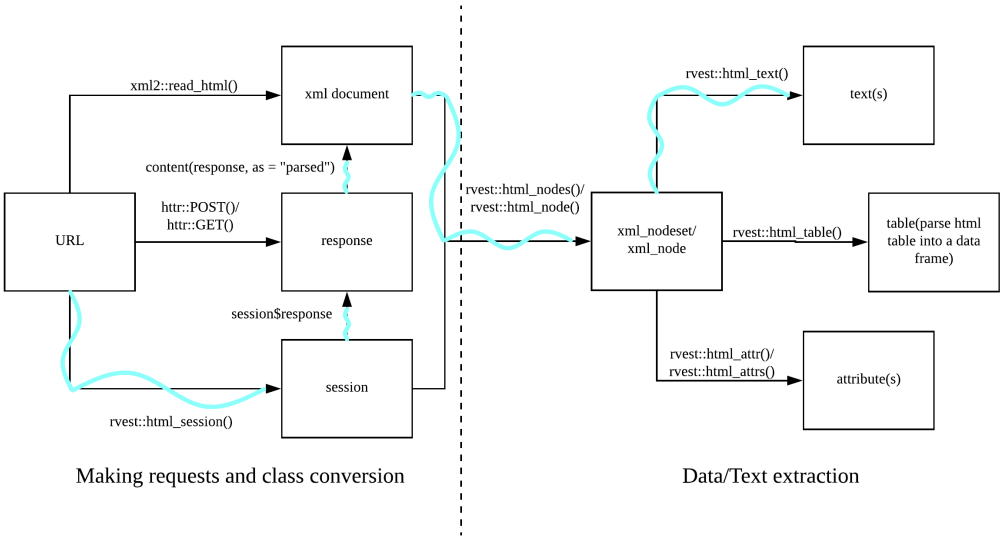
```
.get_delay = function(domain) {

message(sprintf("Refreshing robots.txt data for %s...", domain))
```

```
cd_tmp = robotstxt::robotstxt(domain)$crawl_delay

if (length(cd_tmp) > 0) {
star = dplyr::filter(cd_tmp, useragent=="*")
if (nrow(star) == 0) star = cd_tmp[1,]
as.numeric(star$value[1])
} else {
10L
}

}

get_delay = memoise::memoise(.get_delay)
```

2.4 Scraping with `rvest`

To start a general scraping function it is required a target url. Then a `html_session`, which is the analogue of a GET/POST request, is opened by specifying the url and the request data that the user needs to send to the web server, left part to the dashed line of the image below. Server request are handled by `httr` since it is the R-gold-standard for server GET and POST requests. Information to be required to the server have been already explored in the previous sections, tough they are mainly three: User Agents, emails references and proxy servers. Once the connection is established (request response 200) all the following operations rely on the opened session, in other words for the time being in the session the user will be authorized with the provided characteristics through the request. Once the connection is established the pure data extraction part can begin. One of the major advancements of the structure represented below is iterability along a predefined set of url. Since all the following url can be generated by the first, the structure is self containing

and can be easily applied. What it is worth noting is that inside each url other sub-urls are given. Those urls are the ones corresponding to each single rental advetisement that have to be inspected singularly one by one by the workflow represented below.



What it comes in the right part of the dashed vertical line is a succession of functions that follow a logical step by step path. `rvest` under the hood handles, in this precise order: reading the html structure of the web page within the session `read_html()`, parsing the html content looking for a single node `html_nodes()` and then converting it into readable text `html_text()`. The entire process can be thought as sort of autoencoder by adding decoding layer after layer on top of the starting url, until reaching the final requested node. the reason why the process appear straightforward is a result of the bleding of `rvest` together with the tidyverse pipe `%>%` operator. Below it is shown a function that scrapes the price and composes the API.

```
scrapeprice.imm = function(session) {

opensess = read_html(session) price = opensess %>% html_nodes(css
    =".im-mainFeatures__title") %>% html_text() %>% str_trim()

if(is.null(price) || identical(price, character(0))) { price2 =
```

```
    opensess %>% html_nodes(css ='.im-features__value ,
    .im-features__title') %>% html_text() %>% str_trim()

if ("prezzo" %in% price2) { pos = match("prezzo",price2)
    return(price2[pos+1]) %>% str_replace_all(c("€"="","\\."="")) %>%
    str_extract( "\\-*\\d+\\.*\\d*") %>% str_replace_na() %>%
    str_replace("NA", "Prezzo Su Richiesta") } else {
    return(NA_character_) } } else { return(price) %>%
    str_replace_all(c("€"="","\\."="")) %>% str_extract(
    "\\-*\\d+\\.*\\d*") %>% str_replace_na() %>% str_replace("NA",
    "Prezzo Su Richiesta")

}

}
```

The function takes as a single argument a session which is established in one other function. Then It reads the session storing the information into an obj called the `opensess`. Another obj is created, namely price, right after the pipe operator a css query into the html is called. The css query `.im-mainFeatures__title` points to a precise data stored in immobiliare web page header, right below the main title. Expectation are that price is a one-element chr vector, containing the price and some other unnecessary characters. Then the algorithm enters into the first `if` statement. The handler checks is the object `price` is empty. If it doesn't the algorithm jumps to the end and returns the cleaned quantity. But If it does it takes again the `opensess` and redirect to a second css query `.im-features__value , .im-features__title` where price, once again, could be found. Please note that This is all done within the same session, so no more request information has to be sent. Since the latter css query points to data stored inside a list, for the time being the newly created obj price2 is a list containing various

information. Then the flow enters into the second `if` statement that checks whether `"prezzo"` is in the list or not, if it does it returns the +1 position index element with respect to the "prezzo" positioning. This happens because data in price2 list are stored by couples sequentially, say: [title, "Appartamento Sempione", energy class, "G", "prezzo", 1200/al mese]. When it returns the element corresponding to +1 position index it applies also some data wrangling with `stringr` to keep out overabundant characters. The function then escapes in the else statement by setting `price2 = NA_Character_`. the Character quality has to be imposed due to fact that if the function is evaluated for a url and returns a quantity x and then is evaluated for url2 and outputs NA (no character) then results can not be combined into dataframe.

The skeleton shown can be shared among the other scraping functions, what it changes is the item "prezzo" which is then replaced by other covariates reasearched, say energy class etc. Moreover some other functions need to have heavy cleaning steps in order to be usable. As a consequence functions need also to be broken down into pieces by single .R files, so that they can be adapted with respect to their singular necessities, see the nex paragraph.

Once all the functions have been created they need to be called together and then data coming after them need to be combined. This is done by `get,data.catsing()` which at first checks the validity of the url, then takes the same url as input and filters it as a session object. Then simultaneously all the functions are called and then combined. All this happens inside a `foreach` parallel loop called by `scrape.all.info()`

```
scrape.all.info = function(url =
    "https://www.immobiliare.it/affitto-case...",
vedi = FALSE,
scrivi = FALSE,
silent = FALSE){
if (silent) {
start = as_hms(Sys.time()); cat('Starting the process...\n\n')
message('\nThe process has started in',format(start,usetz = TRUE))
}
# open parallel multisession
cl = makeCluster(detectCores()-1) #using max cores - 1 for parallel
    processing
registerDoParallel(cl)
start = as_hms(Sys.time())
```

```
if (silent) {
message('\n\nStart all the requests at time:', format(start,usetz =
    T))
}
ALL = foreach(i = seq_along(links),
.packages = lista.pacchetti,
.combine = "bind_rows",
.multicombine = FALSE,
.export ="links" ,
.verbose = TRUE,
.errorhandling='pass') %dopar% {
source("utils.R")
sourceEntireFolder("functions_singolourl")
get.data.catsing = function(singolourl){

# dormi()
#
if(!is_url(singolourl)){
stop(paste0("The following url does not seem either to exist or it is
    invalid", singolourl))
}

session = html_session(singolourl, user_agent(agents[sample(1)]))
if (class(session) == "session") {
session = session$response
}

id = tryCatch({scrapehouse.ID(session)},
error = function(e){ message("some problem occured in
    scrapehouse.ID") })
lat = tryCatch({scrapelat.imm(session)},
error = function(e){ message("some problem occured in scrapelat.imm")
    })
long = tryCatch({scrapelong.imm(session)},
error = function(e){ message("some problem occured in
    scrapelong.imm") })
location = tryCatch({take.address(session)},
error = function(e){ message("some problem occured in take.address")
    })
condom = tryCatch({scrapecondom.imm(session)},
error = function(e){ message("some problem occured in
    scrapecondom.imm") })
buildage = tryCatch({scrapeagebuild.imm(session)},
error = function(e){ message("some problem occured in
    scrapeagebuild.imm") })

...

combine = tibble(ID = id,
LAT = lat,
LONG = long,
LOCATION = location,
CONDOM = condom,
BUILDAGE = buildage,

...


return(combine)
}
stopCluster(cl)
return(ALL)
}
```

Below it is printed the tree structure folder that composes the main elements of the scraping procedure. It can be spotted that the two folders, namely functions_singolourl and functions_url enclose all the single functions that allows to grab single information from session. Folders with a customized function are then sourced within the two main functions, scrape.all and scrape.all.info so data can be extracted.

```
 levelName
1  immobiliare.it-WebScraping
2   ¦--functions_singolourl
3   ¦   ¦--0scrapesqfeetINS.R
4   ¦   ¦--0scrapenroomINS.R
5   ¦   ¦--0scrapepriceINS.R
6   ¦   ¦--0scrapetitleINS.R
7   ¦   ¦--ScrapeAdDate.R
8   ¦   ¦--ScrapeAge.R
9   ¦   ¦--ScrapeAgeBuilding.R
10  ¦   ¦--ScrapeAirConditioning.R
11  ¦   ¦--ScrapeAptChar.R
12  ¦   ¦--ScrapeCatastInfo.R
13  ¦   ¦--ScrapeCompart.R
14  ¦   ¦--ScrapeCondom.R
15  ¦   ¦--ScrapeContr.R
16  ¦   ¦--ScrapeDisp.R
17  ¦   ¦--ScrapeEnClass.R
18  ¦   ¦--ScrapeFloor.R
19  ¦   ¦--ScrapeHasMulti.R
20  ¦   ¦--ScrapeHeating.R
21  ¦   ¦--ScrapeHouseID.R
22  ¦   ¦--ScrapeHouseTxtDes.R
23  ¦   ¦--ScrapeLAT.R
```

```
24  ¦   ¦--ScrapeLONG.R
25  ¦   ¦--ScrapeLoweredPrice.R
26  ¦   ¦--ScrapeMetrature.R
27  ¦   ¦--ScrapePhotosNum.R
28  ¦   ¦--ScrapePostAuto.R
29  ¦   ¦--ScrapePropType.R
30  ¦   ¦--ScrapeReaReview.R
31  ¦   ¦--ScrapeStatus.R
32  ¦   ¦--ScrapeTotPiani.R
33  ¦   ¦--ScrapeType.R
34  ¦   °--take_location.R
35  ¦--scrapeALL.R
36  ¦--scrapeALLINFO.R
37  ¦--functions_url
38  ¦   ¦--ScrapeHREF.R
39  ¦   ¦--ScrapePrice.R
40  ¦   ¦--ScrapePrimaryKey.R
41  ¦   ¦--ScrapeRooms.R
42  ¦   ¦--ScrapeSpace.R
43  ¦   °--ScrapeTitle.R
44  ¦--libs.R
45  ¦--utils.R
46  ¦--README.Rmd
47  ¦--README.md
48  °--simulations
49      ¦--rt_match_vs_forloop.R
50      °--runtime_simul.R
```

2.4.1 Parallel Computing

Since many html sessions are opened and within each session many requests are sent computations can take a while. For the sake of the analysis and the app this should not bother the end user because scraping tasks are performed daily and a single day is sufficient amount of runtime. In any case functions are optimized following the criteria stated before. Run time is crucial when dealing with time series and time to market in real estate is very important, this leads to have always fresh data. A way to secure fresh new data is to have lightweight computation on a single machine o heavy computation divided among a bunch of different machines, in this case sessions. All the following runtime examinations are performed on the `scrape.all` functions. A first attempt was using `furrr` package (Vaughan and Dancho, 2018) which enables mapping through a list with the `purrr` , along with a `future` parallel backend. This has shows decent results, but its run time increases when more requests are sent. This leads to a preventive conclusion about the computational complexity: it has to be at least linear. Empirical demonstrations have been made:

```
library(furrr)


vecelaps = c()
start = c()
end = c()
for (i in 1:len(list.of.pages.imm[1:20])) { start[i] = Sys.time()
    cat("\n\n run iteration", i, "over 20 total\n")
    list.of.pages.imm[1:i] %>% furrr::future_map(get.data.caturl,
    .progress = T) %>% bind_rows()


end[i] = Sys.time() vecelaps[i] = end[i] - start[i] }
```

```
furrrmethod = tibble(start,
end,
vettoelaps)

# ggplot (themed) run time meausurament method 1
p = ggplot(furrrmethod,aes(x=1:20, y=vettoelaps)) +
geom_line( color="steelblue") +
geom_point() +
xlab("Num URLS evaluated") +
ylab("run time (in seconds)") +
ggtitle("Run-Time for First method (furrr multisession)") +
stat_smooth(method=lm) +
theme_nicco()
p
```

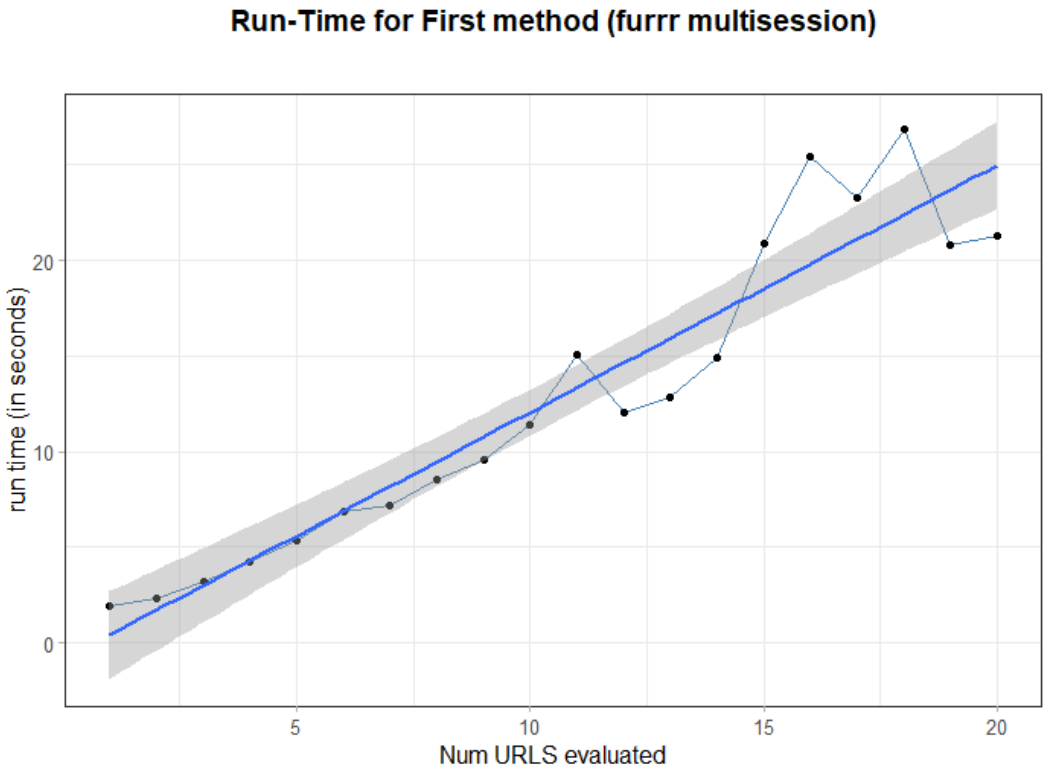


Figure 2.2: computational complexity analysis with Furrr

On the x-axis the number of urls evaluated together, iteration after iteration

the urls considered are increased by one. On the y-axis the time measured in seconds. Looking at the smoothing curve in between an easy guess might be linear time $O(n)$.

A second attempt tried to explore the `foreach` package (Microsoft and Weston, 2020). This interesting package enables a new looping construct for executing R code in an iterative way. With all the variety of existing (`apply`, `lapply`, `sapply`, `eapply`, `mapply`, `rapply`,) looping constructs, it might be doubted that there is a need for yet another construct. The main reason for using the `foreach` package is that it supports *parallel execution*, that is, it can execute those repeated operations on multiple processors/cores on the computer, or on multiple nodes of a cluster. The construction is straightforward:

- start clusters on processors cores
- define the iterator, in this case i = to the elements that are going to be iterated through
- `.packages`: Inherits the packages that are used in the tasks define below
- `.combine`: Define the combining function that bind results at the end (say cbind, rbind or tidyverse::bind_rows). It has to be a string.
- `.errorhandling`: specifies how a task evaluation error should be handle.
- `%dopar%`: the dopar keyword suggests foreach with parallelization method
- then the function within the elements are iterated
- close clusters

One major important thing concerns the fact that th function within iterators repeats itself should be standalone. For standalone it is meant that the body function should be defined inside, as it would be a an empty environment. As a matter of fact packages has to be taken inside each time, and if the function is not defined inside body (or is not source from some other locations) the clusters can not operate and an error is printed.

```
cl = makeCluster(detectCores()-1)
registerDoParallel(cl)


vettoelaps1 = c()
start1 = c()
end1 = c()
for (j in 1:len(list.of.pages.imm[1:20])) {
start1[j] = Sys.time()
cat("\n\n run iteration",j,"over 20 total\n")
foreach(i = seq_along(list.of.pages.imm[1:j]),
.packages = lista.pacchetti,
.combine = "bind_rows",
.errorhandling='pass') %dopar% {
source("main.R")
x = get.data.caturl(list.of.pages.imm[i])
}
end1[j] = Sys.time()
vettoelaps1[j] = end1[j] -start1[j]
}
stopCluster(cl)
```

It can be seen quite easily that the curve is flattened and resembles someway logarithmic time $O(log(n))$.

A further improvement could be obtained using a new package called `doAzureParallel` which is built on top of foreach. doAzureParallel enables different Virtual Machines operates parallel computing throughout Microsoft Azure cloud, but this comes at a substantial monetary cost. This would be a perfect match given that parallel methods seen before accelerates the number of requests sent among different processors or cluster, even though actually what it is really needed it is something that separates session. Unleashing Virtual Machines permits from one hand to further increase computational

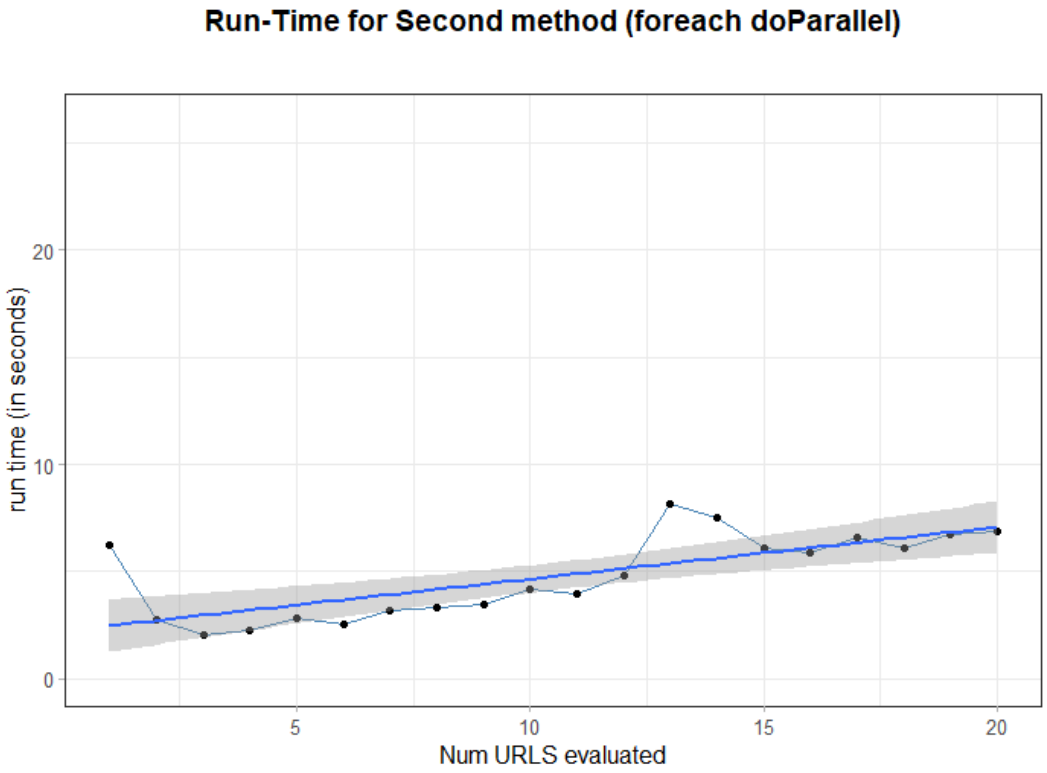


Figure 2.3: computational complexity analysis with Furrr

power and the number of potential requests, from the other it can splits requests among different user agents (a pool for each VM) masquerading even better the scraping automation.

2.5 Further Improvements

The main challenge remains unsolved since each single elements has been finely optimized but API continues to be unstable. What it can not be optimized are the system ad choices to change the layout of the page or to change the url structure (allocation of datain the web page). The way the API is designed really facilitates fast debugging but this can not be automatized. The API continues to necessitate to have a continuous back end review to verify thw working status. Moreover Error messages can not really be undestood sometimes, this is due to functions that are called within a parallel beckend that does not allow [refererenza su stack overflow di errore di print] to print error on

console. So each time an error occur the "main" functions needs to be taken out of from the foreach parallel backend and evaluated in isolation where the loop ends. This is trivial but for the time being no solutions have been found.

2.6 Legal Challenges (ancora non validato)

"Data that are online and public are always available for all" is never a good answer to the question "Can I use that data to my scope". Immobiliare.it[1] is not providing any source of data from its own database neither it is planning to do so in the future. It has not even provided a paid API through which might be possible to perform analysis. However the golden standard for scraping was respected since the robot.txt file is clear allowing any actions as demonstrated above. What it worth noting is that some other popular player other like Idealista is using a different approach. Some of procedures that has been applied to the immboliare scraping was not possibile on the Idealista. This could be due to many reason:

- Idealista content is composed by Javascript so and html parser can no get that.
- Idealista blocks also certain web browser that have a demonstrated "career" in scraping procedures.

All of this leads to accept that entry barriers to scrape are for sure higher than the one faced for Immobiliare. The reticence to share data could be a reflex on how big idealista is; as a matter of fact it has a heavy market presence in some of the Europe real estates country as Spain and France. So what they thought was to raise awareness on scraping procedure that in a certain way can hurt their business. This has been validated by the fact that prior filtering houses on their website a checkbox has to be signed. The checkbox make the user sign an agreement on their platform according to which data can not be misused and it belongs their intellectual property.

[1] https://www.immobiliare.it/

Chapter 3

Infrastructure

In order to provide a fast, portable and integrated product to the end user It has been designed a quite straightforward software architecture. We have already seen the scraping functions and how they are built around the concept of easy flexibility and debugging. This is due to the fact that they should extract something that is dynamic, it is not sure that it will be as the day before. The data we are trying to grab might have been moved somewhere else throughout the website. Or it might have placed extra expression inside the node we are inspecting ("$" sign following the monthly rental price) . A very often occurring example regards the way information concerning the house are represented in the website. Considering the september 2018 january 2020 time span the design of the website has changed a vast number of times. Since both the design and the scrapping functions relies on the HTML skelethon and CSS queries. As soon as something changes in the website the other files needs to be readjusted to be consistent with the content and so back and forth. The debugging handlers nested in the functions helps the maintainer to grasp what it is not working properly where the error occur. The same inner philosophy has been applied to the software architecture chosen for this project. First of all the wide range of open source solutions (back-end and front-end) and documentation on this has made many analyst and data scientist almost full stack developer. This was also due to the fact that RStudio has set very well

oriented guidelines spending a lot of effort giving its users an easy, integrated and interconnected environment. By that it is meant that recently the RStudio community has developed, on top of many different others, an entire package dedicated to REST APIs (Plumber (Trestle Technology, LLC, 2018)). MOreover developers in RStudio and its contributors have created an entire new paradigm called Shiny (Chang et al., 2020), a popular web app development package, that forces the user to have front-end and back-end technologies tied up in the same IDE (RStudio) and with a unique language to deal with. The front end file (for simplicity named UI.R) contains the UI's layout and the style and also other javascripts components. On the other hand the server file (named after server.R) absorbs the back-end, under the hood, code and makes the UI intercact and respond to the user. This comes at a cost of flexibility and customization since Shiny could not easily handle too many embellishments (even though potentially can). Nevertheless a unique environment makes integration with other technologies easier and most of all introduces the analyst to a full stack approach. Many open source projects are gravitating around the Shiny framework with the aim to extend its capabilities. One example is a newly created package called reactR (Inc et al., 2020) that allows user to implement the power of React.js into the shiny UI front end. All of this is possible, once again, by the R community but a greater contribution come from digging up the right path along which everything by open source comes natural. (parallelo con la vigna e l'albero che la sostiene e indirizza) The carrier idea for this project is to have parallelized scraping functions called daily by a scheduler producing and subsequently storing a .csv file in a MySQL /cartoDB database. They are all tought to be containerized in a Linux (Ubuntu distr) docker container hosted in a AWS EC2 server. Then in a second container a Shiny app is placed, this one pipes in data from the former infrastructure and apply the statistical model stored (by an API call) in its server.R part.

The main technologies implied are:

- Scheduler cron job

- Docker containers
- Shiny
- Plumber REST API
- AWS (Amazon Web Services) EC2
- CartoDB

On top of that even each single part of this thesis has been made stand alone and can be easily accessed and modified through this link[1]. The pdf (theis) version of the gitbook can be obtained by clicking the download button that can be seen in figure below. A Latex engine (Xelatex) wrapped into the website compiles a sequence of Markdown documents converting them into .html (the book's chapters) which are formatted by rules grouped in a .yml file. All the documents are pushed to a Github repository with git. By a simple trick, since all the files are static html, they can be displayed through GH pages as it is a website. All of this has been possible thanks to Bookdown (Xie, 2020) once again a R well documented package (Xie, 2016) to build interactive books along with RMarkdown (Allaire et al., 2020).

An empirical observation of immobiliare.it has suggested that houses rents advertisement are continuously added and then removed during the day. Fresh data is needed to have updated analysis since the scope in here is to offer realtime considerations. Something should be automated periodically in order to address the issue. Moreover, as rule of thumb, a daily data extraction might be a good option for some reasons. It can intercept price variations with a relatively small time lag, It can also display some sort of pattern in time that would help the reader/user to select the perfect choice. As a consequence a daily .csv file is generated and directly collected into a Db folder arranged by time The solutiond proposed takes care of the issue by making the scraping script generating the .csv be executed by a scheduler.

[1] https://niccolosalvini.github.io/Thesis/

3.1 Scheduler

A Scheduler in a process is a component on a OS that allows the computer to decide which activity is going to be executed. In the context of multi-programming it is thought as a tool to keep CPU occupied as much as possible. As an example it can trigger a process while some other is still waiting to finish. There are many type of scheduler and they are based on the frequency of times they are executed considering a certain closed time neighbor.

- Short term scheduler: it can trigger and queue the "ready to go" tasks
  - with pre-emption
  - without pre-emption

The ST scheduler selects the process and It gains control of the CPU by the dispatcher. OIn this context we can define latency as the time needed to stop a process and to start a new one.

- Medium term scheduler
- Long term scheduler

for some other useful but beyond the scope information, such as the scheduling algorithm the reader can refer to (Wikiversità, 2020).

The scheduler in this context cosists in a .sh (shell file, sort of text file) composed by a set of instructions that are being executed by the computer on daily basis. This file has to be in the same WD (working directory) of the project in order to make it working. Some common issues can occur when new files coming after the execution of the scheduled main script are generated, but the path isnt explicitly specified. This can lead to the partial or incomplete generation of the file since the shell file is executed within the folder but is triggered by some other location on the computer. Each OS has its own scheduler and syntax to call it. Since we are interested in Ubuntu machines the scheduler

is said to be a cron job. Later it will be clear why Ubuntu is the option to pursue.

va parafrasato

3.1.1 Cron Jobs

The software utility cron also known as cron job is a time-based job scheduler in Unix-like computer operating systems. Users that set up and maintain software environments use cron to schedule jobs (commands or shell scripts) to run periodically at fixed times, dates, or intervals. It typically automates system maintenance or administration—though its general-purpose nature makes it useful for things like downloading files from the Internet and downloading email at regular intervals.

The actions of cron are driven by a crontab (cron table) file, a configuration file that specifies shell commands to run periodically on a given schedule. The crontab files are stored where the lists of jobs and other instructions to the cron daemon are kept. Users can have their own individual crontab files and often there is a system-wide crontab file (usually in /etc or a subdirectory of /etc) that only system administrators can edit.

Each line of a crontab file represents a job, and looks like this:

```
# ┌──────────── minute (0 - 59)
# │ ┌────────── hour (0 - 23)
# │ │ ┌──────── day of the month (1 - 31)
# │ │ │ ┌────── month (1 - 12)
# │ │ │ │ ┌──── day of the week (0 - 6) (Sunday to Saturday;
# │ │ │ │ │                                7 is also Sunday on some systems)
# │ │ │ │ │
# │ │ │ │ │
# * * * * * <command to execute>
```

Figure 3.1: crontab

Each line of a crontab file represents a job. This example runs a shell program called scheduler.sh at 23:45 (11:45 PM) every Saturday.

45 23 * * 6 /home/oracle/scripts/scheduler.sh

Some rather unusual scheduling definitions and syntax for cronjobs can be found in this reference (Wikipedia contributors, 2020)

The cron job applied to the script needs to be ran at 11:30 PM everyday. It has that forms: —> qui immagine

va parafrasato

For now the computational power comes from the machine on which the system is installed. A smarter solution takes into consideration that the former infrastructure has its own limits. Major limits comprehend run time since at the same moment the machine runs locally both the scraping functions and the app computations. This to a certain extent might fit for personal use but as data increases all the system risks to fail. It is also totally local so the analysis can not be shared with anyone. This problem can be addressed with a technology that has seen a huge growth in its usage in the last few years: Docker containers.

3.2 Docker Container

from docker In 2013, Docker introduced what would become the industry standard for containers. A container is a standard unit of software that packages up code and all its dependencies so the application runs quickly and reliably from one computing environment to another. A Docker container image is a lightweight, standalone, executable package of software that includes everything needed to run an application: code, runtime, system tools, system libraries and settings.

3.2.1 What is Docker?

Container images become containers at runtime and in the case of Docker containers - images become containers when they run on Docker Engine. Available for both Linux and Windows-based applications, containerized software will al-

ways run the same, regardless of the infrastructure. Containers isolate software from its environment and ensure that it works uniformly despite differences for instance between development and staging.

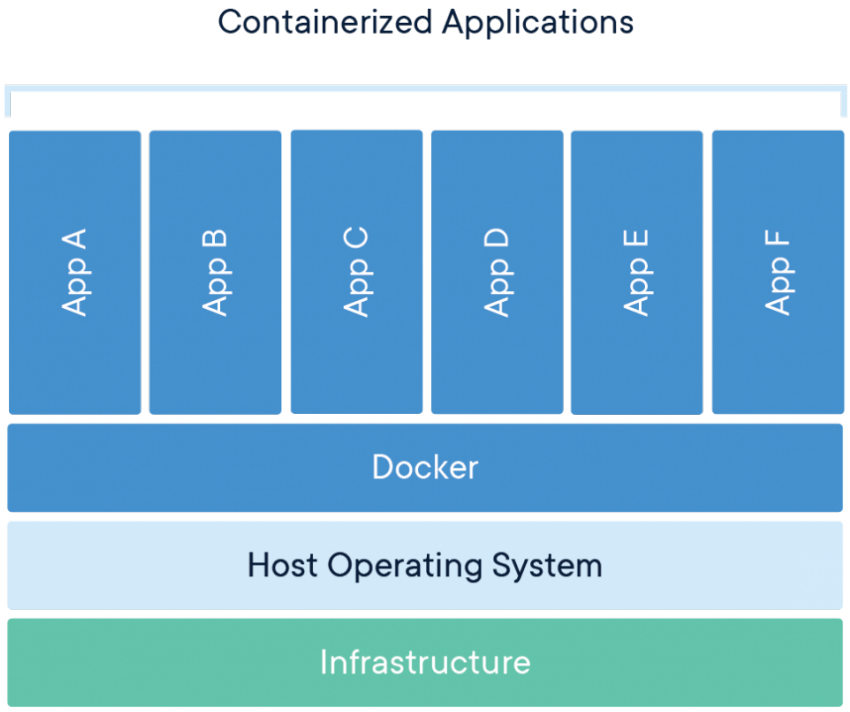


Figure 3.2: docker example

Docker leveraged existing computing concepts around containers and specifically in the Linux world. Docker's technology is unique because it focuses on the requirements of developers and systems operators to separate application dependencies from infrastructure.

A question might come up about why a Virtual Machine could not be a preferable container for our specified task. Well, Containers and virtual machines have similar resource isolation and allocation benefits, but function differently because containers virtualize the operating system instead of hardware. Containers are more portable and efficient.

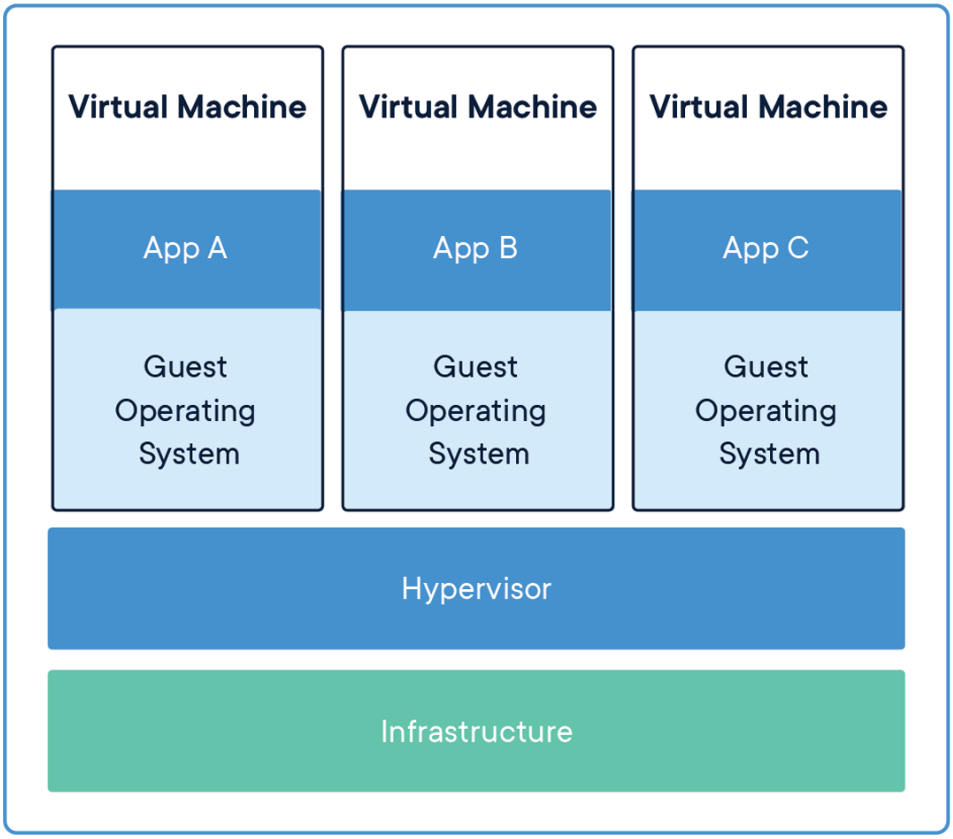


Figure 3.3: docker container vs VM

from docker

3.2.2 What are the main andvantages of using Docker

va parafrasato from Matt Dancho

Indeed, the popular employment-related search engine, released an article this past Tuesday showing changing trends from 2015 to 2019 in "Technology-Related Job Postings". We can see a number of changes in key technologies - One that we are particularly interested in is the 4000% increase in Docker.

Top 20 tech skills in 2019

Percent of all tech jobs, change September 2014 to September 2019

Rank	Skill	2014 share	2019 share	% change
1	sql	23.6%	21.9%	-7%
2	java	19.7%	20.8%	6%
3	python	8.1%	18.0%	123%
4	linux	14.9%	14.9%	0%
5	javascript	12.4%	14.5%	17%
6	aws	2.7%	14.2%	418%
7	c++	10.6%	10.7%	1%
8	c	9.3%	10.3%	11%
9	c#	8.3%	9.3%	11%
10	.net	9.9%	8.4%	-15%
11	oracle	13.5%	8.4%	-38%
12	html	9.8%	8.1%	-17%
13	scrum	4.8%	8.0%	64%
14	git	3.1%	7.8%	148%
15	css	7.8%	7.3%	-5%
16	machine learning	1.3%	7.0%	439%
17	azure	0.6%	6.9%	1107%
18	unix	10.0%	6.7%	-33%
19	sql server	7.8%	6.5%	-17%
20	docker	0.1%	5.1%	4162%

Source: Indeed

indeed

Figure 3.4: docker-stats

The landscape of Data Science is changing (was previously an Economist at the Indeed Hiring Lab, 2020) from reporting to application building:

In 2015 - Businesses need reports to make better decisions In 2020 - Businesses need apps to empower better decision making at all levels of the organization This transition is challenging the Data Scientist to learn new technologies to stay relevant…

As a matter of fact, it is no longer sufficient to just know machine learning algorithms. Future data workers need to know how to put machine learning into production as quickly as possible to meet the business needs. This can be done either integrating existing obsolete/old technologies with the new ones, or build a solid, portable and scalable infrastructure to better the processes. In order to do so, It is strictly needed to learn from programmers the basics of Software Engineering. The extremely good news is that no data scientist whatsoever needs to be a perfect software engineer, but at least he has to know how to integrate already built techonlogies with its work. This truly can help in the quest to unleash data science at scale and unlock real business value. The way Docker does that are many (red, 2020):

- *Rapid application deployment* : containers include the minimal run time requirements of the application, reducing their size and allowing them to be deployed quickly.
- *Portability across machines* :an application and all its dependencies can be bundled into a single container that is independent from the host version of Linux kernel, platform distribution, or deployment model. This container can be transfered to another machine that runs Docker, and executed there without compatibility issues.
- *Version control and component reuse* : you can track successive versions of a container, inspect differences, or roll-back to previous versions. Containers reuse components from the preceding layers, which makes them noticeably lightweight.
- *Sharing* : you can use a remote repository to share your container with others. It is also possible to configure a private repository hosted on Docker Hub.

- *Lightweight footprint and minimal overhead* : Docker images are typically very small, which facilitates rapid delivery and reduces the time to deploy new application containers.
- *Simplified maintenance* :Docker reduces effort and risk of problems with application dependencies.

The way all of this is possible is a dockerfile that determines the instruction that docker has to perform to abstract the environment.

3.2.3 Dockerfile

Docker can build images automatically by interpreting the instructions from a Dockerfile. A Dockerfile can be thought as a written recipe to cook a specific cake, with all the ingredients described in a piece of a paper using a generic oven and adding layer of preparation after layer. A Dockerfile is a text format document that contains all the commands/rules a generic user could call on the CLI to assemble an image. Executing the command `docker build` from shell the user can trigger the image building. That executes several command-line instructions in chronological succession of steps. The Dockerfile used to trigger the build of the docker image has this following set of instructions:

- `FROM rocker/r-ver:4.0.0` : the command imports an image already written by the rocker team (authored contributors for the R docker project) that contains the base-R version 4.0.0. Recently with the 4.0 version the RStudio team has created a repository management server for its packages that organizes and centralizes R packages (offline access and checkpoints). This will shorten the installation time and secure packages since they all can be freezed into a version that make the whole system works.

- `RUN R -e "install.packages(c('plumber','tibble','...',dependencies=TRUE)` : the command install all the packages required to execute the files (R

Figure 3.5: dockerfile

files) containerized for the scraping. Since all the packages have their dependencies the option `dependencies=TRUE` is needed.

- `EXPOSE 8000` : the commands instructs Docker that the container listens on the specified network ports 8000 at runtime. It is possible to specify whether the port exposed listens on UDP or TCP, the default is TCP (this part needs a previous set up previous installing, for further online documentation It is reccomended (doc, 2020))

- `ENTRYPOINT ["Rscript", "main.R"]` : the command tells docker to execute the Rscript extension file main.R within the container that triggers the API building/the generation of the .csv file.

va parafrasato from Matt Dancho

An alternative and very used approach could be wrapping all the scraping function into an API and then send a `GET` request to the API endpoint needed.

3.3 API

va parafrasato

The scraping functions, according to how the author as structured them, are able to produce two .csv extension (if the boolean option `write` is set = TRUE) files. As already clarified in the previous ssection some point should be joined by a primary key. But for the sake of In order to give the possibility to have a daily updated saptial analysis on data we need to continously have fresh data. In the website data come and go, as products in a marketplace, so the main idea is to have something that catches the new added and deletes what it is already taken. Nowadays we have many open source, nearly cost free, techonlogies that allow us to have corporate grade applications that can be orizontally scaled at need. Most of them come with great docuemntation and ready to use examples that flatten the learning curve. The first choice that has to be made is: either to provide a .csv file day by day with all the data to feed the application, or we exploit some portable and fast solutions as API.

va parafrasato

3.3.1 What is an API

API is a set of definitions and protocols for building and integrating application software. API stands for application programming interface.

APIs let a product or a service communicate with other products and services without having to know how they're implemented. This can simplify app development, saving time and money. When you're designing new tools and products—or managing existing ones—APIs give flexibility; simplify design, administration, and use; and provide opportunities for innovation. APIs are sometimes thought of as contracts, with documentation that represents an agreement between parties: If party 1 sends a remote request structured a particular way, this is how party 2's software will respond. APIs in less formal verbs are infrastructure that calls function with given arguments and answer

back with elaborations of the data passed. They are expressed through link and could have different end points. An end point identifies the operation that the API caller wants to perform. The most of the times APIs are protected with encryption and authentication.

API examples: - Google Maps API: allows developers embed geo-location data using JavaScript. The Google Maps API is designed to work on mobile and desktop. - YouTube API: allows developers integrate YouTube videos and functionalities into websites or applications. - Google Analytics API: allows to track website performance in terms of audience, monetization and other important metrics throught the Google Analytics interface. The website the thesis come from has this implementation working.

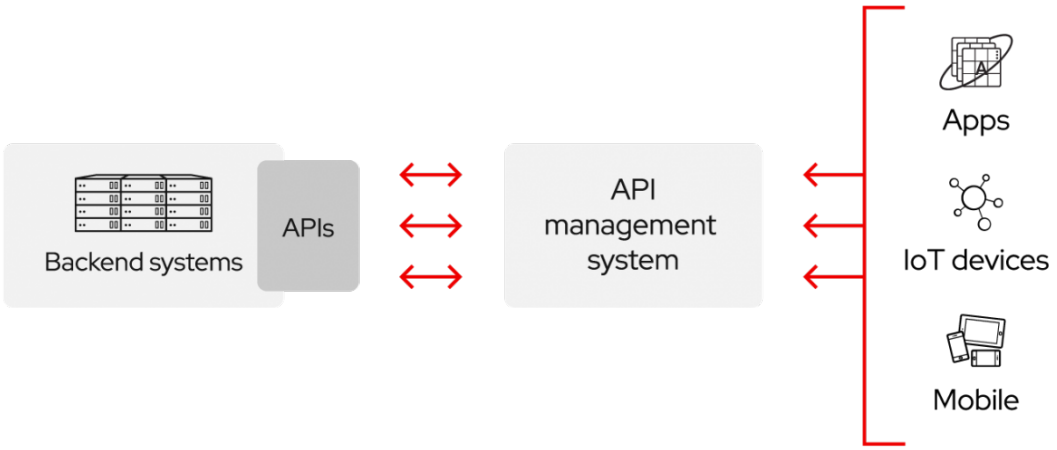


Figure 3.6: API functioning

Because APIs simplify how developers integrate new application components into an existing architecture, they help business and IT teams collaborate. Business needs often change quickly in response to ever shifting digital markets, where new competitors can change a whole industry with a new app. In order to stay competitive, it's important to support the rapid development and deployment of innovative services. Cloud-native application development is an identifiable way to increase development speed, and it relies on connecting a microservices application architecture through APIs.

3.3.2 Plumber API

Plumber (api, 2020) allows the user to create a web API by simply adding decoration comments to the existing R code. Decorations are a special type of comments that suggests the plumber where and when the API parts are. Here below a toy example from the reference:

```
# plumber.R file

# * Echo back the input * @param msg The message to echo * @get /echo
function(msg = "") {
    list(msg = paste0("The message is: '", msg, "'"))
}

# * Plot a histogram * @png * @get /plot
function() {
    rand = rnorm(100)
    hist(rand)
}

# * Return the sum of two numbers * @param a The first number to add * @param
# The second number to add * @post /sum
function(a, b) {
    as.numeric(a) + as.numeric(b)
}
```

This chunk of code assembled into a .R file has three endpoints which can be easily recognized by the number of functions present.

Special comments are marked as this `#*` and they are followed by specific keywords denoted with `@`. - the name of the API (unique without the `@`) - the `@params` keyword refers to parameter that has to be inputted to give the result of the function define below. If in the function below defualt parameters are

stated then the API response is the elaboration of the functions with those parameters. If the function does not specify any parameter, see the below endpoints below, plumber has to make sure that the function can run without them. - the `#* @png` chuck piece specify the extension of the output file. - the `#* @get /plot` decorations specify the type of HTTP request we are sending, in this case a GET request. The user in this case is requesting some information to the API, the name is plot, so the expectations are that a plot is given as response. - the `@filter` decorations activates a filter through which the maintainer can track users logs data. Filters can also handle data coming from users formatting it to the standard requested.

3.3.3 Immobiliare API design

Daily extractions give birth to a more or less constant number of total rows which are 3200 - 3500 and a constant number of covariates 52 (the parameter npages can handle the number of pages to be scrapped). Data can come from two different end-points */scrape* and */complete* as the API description says. This is due to the fact that the former end point response is faster than the latter, since it requires fewer sessions opened. In the below figure it can be seen that within a single session (same website page) the user can grab data from max 25 different rental offers, this is what the end point /scrape does. The latter as opposite goes inside each single sub-link and pick up all the information/covariates through a specified set of functions, this is what /complete does. The other reason why it takes more run time relies on the fact that the links of each of the rental advertisements are not generated in local, like the one in the /complete. At first In order to scrape each specific single advertisement through its respective url they need to be extracted. This is done by the `all.links()` function which corresponds also to the second end point /links.

- Get fast raw data, 5 covariates: title, price, num of rooms, sqmeter, primarykey

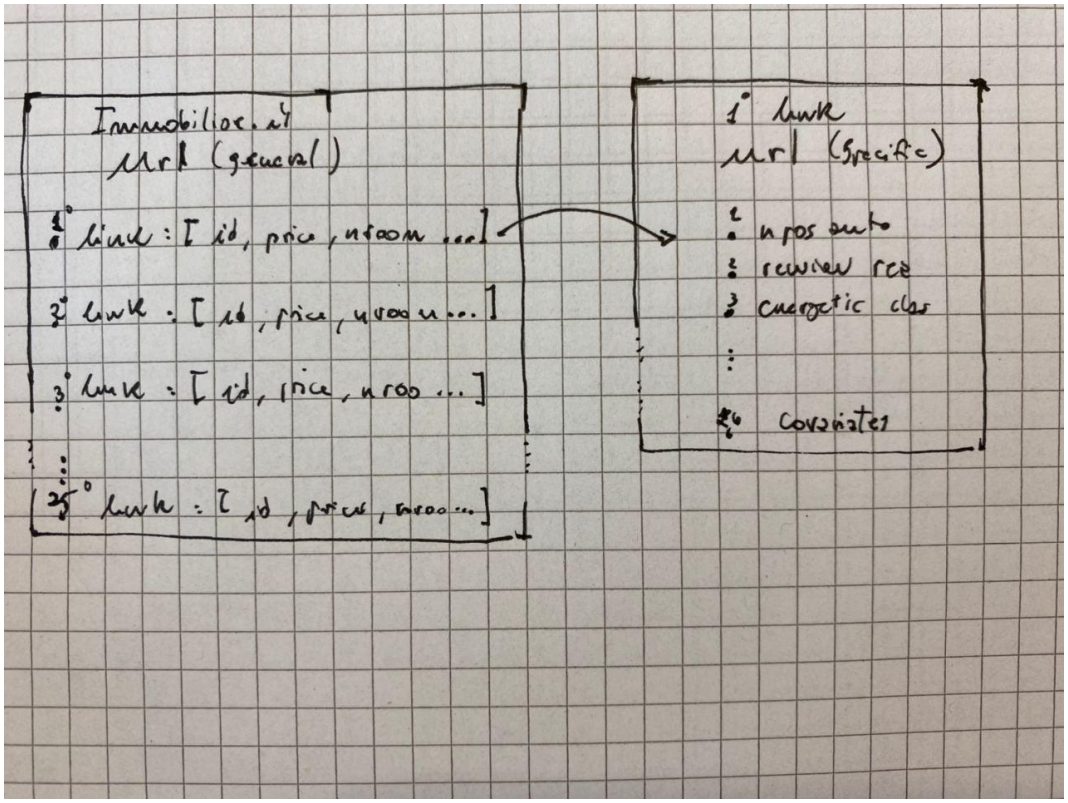


Figure 3.7: Endpoints Architecture

```
GET */scrape

param url [url string] the link from which you are interested to extract
param npages [positive integer] number of pages to scrape (1-300)
content-type: application/json
```

- Get all the links

```
GET */link

param url [url string]  the link from which you are interested to extrac
param npages  [positive integer] number of pages to scrape (1-300)
content-type: application/json
```

- Get the complete set of covariates (52) from each single links, takes a while

```
GET */complete


param url  _url string_ url the link from which you are interested to ex
param npages [positive integer] number of pages to scrape (1-300)
content-type: application/json
```

3.4 AWS EC2 server

The structure needs to be hosted on a server that permits to scale computation and manage multiple requests.

from amazon Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides secure, resizable compute capacity in the cloud. It is designed to make web-scale cloud computing easier for developers. It provides complete control of the computing resources and let run on Amazon's proven computing environment. **from amazon**

These are virtual servers that you can spin up and rent. You can set the servers up however you want, and you can scale them up or down (to provide more juice) as needed.

(…)

posso far vedere: - come si fa deplyment su AWS quindi - inizializzazione istanza - settare paramtri - mettere spazio macchina - spiegare perchè conviene tenere un server AWS

Chapter 4

Data Presentation

Data come packed into an API in JSON format and are ready to be analysed. It has 52 covariates and the number of rows rows strictly depends on the API query parater settings.

4.1 Data Glimpse

```
library(knitr)
glimpse(data)
```

```
## Rows: 250
## Columns: 34
## $ X                         <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,
## $ ID                        <int> 82585523, 82710269, 80532997, 80880239, 82
## $ LAT                       <dbl> 45.4770, 45.4709, 45.4605, 45.4709, 45.470
## $ LONG                      <dbl> 9.15101, 9.20868, 9.18927, 9.20868, 9.2086
## $ LOCATION                  <chr> "piazzale arduino C.A.", "via antonio kram
## $ CONDOM                    <int> 310, 117, 417, 117, 133, 170, 333, 317, 35
## $ BUILDAGE                  <int> 1940, 1920, 1967, 1920, 1920, 1990, 1900,
## $ FLOOR                     <chr> "6°, con ascensore", "1° piano", "4°, con
```

```
## $ INDIVSAPT                <chr> "Appartamento", "Appartamento", "Appartame
## $ LOCALI                   <chr> "3 (2 camere da letto, 1 altro), 2 bagni,
## $ TPPROP                   <chr> "Residenziale", "Residenziale", "Residenzi
## $ STATUS                   <chr> "Nuovo / In costruzione", "Ottimo / Ristru
## $ HEATING                  <chr> "Centralizzato, a radiatori, alimentato a
## $ AC                       <chr> "Autonomo, freddo", NA, "Autonomo, freddo"
## $ PUB_DATE                 <chr> "2020-09-22", "2020-09-22", "2020-09-24",
## $ CATASTINFO               <chr> "Classe A/3, rendita \200 697", "Classe A/
## $ APTCHAR                  <chr> "- - fibra ottica- - - videocitofono- - -
## $ PHOTOSNUM                <int> 20, 20, 20, 20, 20, 20, 20, 20, 16, 20, 18
## $ AGE                      <chr> "Skyline RE Milano", "Skyline RE Milano",
## $ LOWRDPRICE.originalPrice <chr> NA, NA, NA, "\200 1.400", NA, NA, NA, NA,
## $ LOWRDPRICE.currentPrice  <chr> NA, NA, NA, "\200 1.300", NA, NA, NA, NA,
## $ LOWRDPRICE.passedDays    <int> NA, NA, NA, 8, NA, NA, NA, NA, NA, NA, NA,
## $ LOWRDPRICE.date          <int> NA, NA, NA, 8, NA, NA, NA, NA, NA, NA, NA,
## $ ENCLASS                  <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA
## $ CONTR                    <chr> "Affitto", "Affitto", "Affitto", "Affitto"
## $ DISP                     <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA
## $ TOTPIANI                 <chr> "7 piani", "5 piani", "4 piani", "5 piani"
## $ PAUTO                    <chr> NA, NA, NA, NA, NA, NA, NA, NA, "1 in gara
## $ REVIEW                   <chr> "Rif: CITY LIFE CUBE - OK CONTRATTO SOCIET
## $ HASMULTI                 <lgl> TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE,
## $ PRICE                    <int> 2450, 1200, 3000, 1300, 1700, 3850, 5000,
## $ SQFEET                   <int> 110, 82, 180, 82, 82, 157, 208, 160, 90, 4
## $ NROOM                    <int> 3, 2, 5, 2, 3, 5, 5, 4, 2, 2, 1, 3, 2, 3,
## $ TITLE                    <chr> "Trilocale piazzale ARDUINO, Milano", "Bil
```

4.2 Explorative Analysis

4.2.1 Semivariogram Covariogram

[en passant anche isotropy and anisotropy. Con stazionarietà forte e debole]

4.3 Gaussian random fields

Chapter 5

INLA computation

va parafrasatoo

Several types of models are used with spatial and spatio-temporal data, depend- ing on the aim of the study. If we are interested in summarizing spatial and spatio-temporal variation between areas using risks or probabilities then we could use statistical methods like disease mapping to compare maps and identify clusters. Moran Index is extensively used to check for spatial autocorrelation (Moran, 1950), while the scan statistics, implemented in SaTScan (Killdorf, 1997), has been used for cluster detection and to perform geographical surveillance in a non-Bayesian approach. The same types of models can also be used in studies where there is an aetiological aim to assess the potential effect of risk factors on outcomes. A different type of study considers the quantification of the risk of experienc- ing an outcome as the distance from a certain source increases. This is typically framed in an environmental context, so that the source could be a point (e.g., waste site, radio transmitter) or a line (e.g., power line, road). In this case, the meth- ods typically used vary from nonparametric tests proposed by Stone (1988) to the parametric approach introduced by Diggle et al. (1998). In a different context, when the interest lies in mapping continuous spatial (or spatio-temporal) variables, which are measured only at a finite set of specific points in a given region, and in predicting their values at unobserved locations, geostatis-

tical methods – such as kriging – are employed (Cressie, 1991; Stein, 1991). This may play a significant role in environmental risk assessment in order to identify areas where the risk of exceeding potentially harmful thresholds is higher. Bayesian methods to deal with spatial and spatio-temporal data started to appear around year 2000, with the development of Markov chain Monte Carlo (MCMC) simulative methods (Casella and George, 1992; Gilks et al., 1996). Before that the Bayesian approach was almost only used for theoretical models and found little applications in real case studies due to the lack of numerical/analytical or simula- tive tools to compute posterior distributions. The advent of MCMC has triggered the possibility for researchers to develop complex models on large datasets without the need of imposing simplified structures. Probably the main contribution to spatial and spatio-temporal statistics is the one of Besag et al. (1991), who developed the Besag–York–Mollié (BYM) method (see Chapter 6) which is commonly used for disease mapping, while Banerjee et al. (2004), Diggle and Ribeiro (2007) and Cressie and Wikle (2011) have concentrated on Bayesian geostatistical models. The main advantage of the Bayesian approach resides in its taking into account uncertainty in the estimates/predictions, and its flexibility and capability of dealing with issues like missing data. In the book, we follow this paradigm and introduce the Bayesian philosophy and inference in Chapter 3, while in Chapter 4 we review Bayesian computation tools, but the reader could also find interesting the follow- ing: Knorr-Held (2000) and Best et al. (2005) for disease mapping and Diggle et al. (1998) for a modeling approach for continuous spatial data and for prediction

va parafrasatoo

5.1 INLA

For many years, Bayesian inference has relied upon Markov chain Monte Carlo methods (Gilks et al. 1996; Brooks et al. 2011) to compute the joint posterior

distribution of the model parameters. This is usually computationally very expensive as this distribution is often in a space of high dimension.

Havard Rue, Martino, and Chopin (2009) propose a novel approach that makes Bayesian inference faster. First of all, rather than aiming at estimating the joint posterior distribution of the model parameters, they suggest focusing on individual posterior marginals of the model parameters. In many cases, marginal inference is enough to make inference of the model parameters and latent effects, and there is no need to deal with multivariate posterior distributions that are difficult to obtain. Secondly, they focus on models that can be expressed as latent Gaussian Markov random fields (GMRF). This provides the computational advantages (see Rue and Held 2005) that reduce computation time of model fitting. Furthermore, Havard Rue, Martino, and Chopin (2009) develop a new approximation to the posterior marginal distributions of the model parameters based on the Laplace approximation (see, for example, MacKay 2003). A recent review on INLA can be found in Rue et al. (2017).

5.2 Laplace Approximation

An alternative approach to the simulation-based MC integration is analytic approx- imation with the Laplace method. Suppose we are interested in computing the following integral:

$$\int f(x)\mathrm{d}x = \int \exp(\log f(x))\mathrm{d}x$$

where $f(x)$ is the density function of a random variable X. We represent $logf(x)$ by means of a Taylor series expansion evaluated in x = x0:

$$\log f(x) \approx \log f\left(x_0\right) + \left(x - x_0\right) \frac{\partial \log f(x)}{\partial x}\bigg|_{x=x_0} + \frac{\left(x - x_0\right)^2}{2} \frac{\partial^2 \log f(x)}{\partial x^2}\bigg|_{x=x_0}$$

If x0 is set equal to the mode $x* = argmax$, log f(x) then log f(x) $\frac{\partial \log f(x)}{\partial x}\Big|_{x=x^*} = 0$ and the approximation becomes

$$\log f(x) \approx \log f\left(x^*\right) + \frac{\left(x - x^*\right)^2}{2} \frac{\partial^2 \log f(x)}{\partial x^2}\bigg|_{x=x^*}$$

The integral of interest is then approximated as follows:

$$\int f(x)\mathrm{d}x \approx \int \exp\left(\log f\left(x^*\right) + \frac{\left(x - x^*\right)^2}{2} \frac{\partial^2 \log f(x)}{\partial x^2}\bigg|_{x=x^*}\right) \mathrm{d}x$$

$$= \exp\left(\log f\left(x^*\right)\right) \int \exp\left(\frac{\left(x - x^*\right)^2}{2} \frac{\partial^2 \log f(x)}{\partial x^2}\bigg|_{x=x^*}\right) \mathrm{d}x$$

where the integrand can be associated with the density of a Normal distribution. In fact, by setting

$$\sigma^{2*} = -1/ \frac{\partial^2 \log f(x)}{\partial x^2}\bigg|_{x=x^*}$$

we obtain:

$$\int f(x)\mathrm{d}x \approx \exp\left(\log f\left(x^*\right)\right) \int \exp\left(-\frac{\left(x - x^*\right)^2}{2\sigma^{2*}}\right) \mathrm{d}x$$

where the integrand is the kernel of a Normal distribution with mean equal to x$*$ and variance σ^{2*}. More precisely, the integral evaluated in the interval (α, β) is approximated by:

$$\int_\alpha^\beta f(x)\mathrm{d}x \approx f\left(x^*\right) \sqrt{2\pi\sigma^{2*}}(\Phi(\beta) - \Phi(\alpha))$$

where $\Phi(\cdot)$ denotes the cumulative density function of th $Normal(x_i, \sigma^{2*})$ distri- bution.

qui volendo un esempio fatto da me

5.3 The Class of Latent Gaussian Models

The first step in defining a latent Gaussian model within the Bayesian framework is to identify a distribution for the observed data y = (y1, … , yn). A very general approach consists in specifying a distribution for yi characterized by a parameter i (usually the mean E(yi)) defined as a function of a structured additive predictor i through a link function g( ), such that g(i) = i. The additive linear predictor i is defined as follows:

$$\eta_i = \beta_0 + \sum_{m=1}^{M} \beta_m x_{mi} + \sum_{l=1}^{L} f_l\left(z_{li}\right)$$

Here 0 is a scalar representing the intercept; the coefficients $\beta = \{\beta_1, \ldots, \beta_M\}$ quantify the (linear) effect of some covariates $x = (x_1, \ldots, x_M)$ on the response; and $f = \{f_1(\cdot), \ldots, f_L(\cdot)\}$ is a collection of functions defined in terms of a set of covariates $z = (z_1, ..., z_L)$. The terms $f_l(\cdot)$ can assume different forms such as smooth and

nonlinear effects of covariates, time trends and seasonal effects, random intercept and slopes as well as temporal or spatial random effects. For this reason, the class of latent Gaussian models is very flexible and can accomodate a wide range of models ranging from generalized and dynamic linear models to spatial and spatio-temporal models (see Martins et al., 2013 for a review). We collect all the latent (nonobservable) components of interest for the inference in a set of parameters named defined as $\theta = \{\beta_0, \beta, f\}$. Moreover, we denote with $\psi = \{\psi_1, \ldots, \psi_K\}$ the vector of the K hyperparameters. By assuming conditional independence, the distribution of the n observations (all coming from the same distribution family) is given by the likelihood

$$p(y \mid \theta, \psi) = \prod_{i=1}^{n} p\left(y_i \mid \theta_i, \psi\right)$$

where each data point yi is connected to only one element i in the latent field θ. Martins et al. (2013) discuss the possibility of relaxing this assumption

assuming that each observation may be connected with a linear combination of elements in θ; moreover, they take into account the case when the data belong to several distri- butions, i.e., the multiple likelihoods case.

We assume a multivariate Normal prior on with mean and precision matrix $Q(\psi)$, i.e., $\theta \sim \text{Normal}\left(\mathbf{0}, Q^{-1}(\psi)\right)$ with density function given by

$$p(\theta \mid \psi) = (2\pi)^{-n/2}|Q(\psi)|^{1/2} \exp\left(-\frac{1}{2}\theta' Q(\psi)\theta\right)$$

where | | denotes the matrix determinant and is used for the transpose operation. The components of the latent Gaussian field are supposed to be conditionally independent with the consequence that $Q(\psi)$ is a sparse precision matrix.8 This specification is known as Gaussian Markov random field (GMRF, Rue and Held, 2005). Note that the sparsity of the precision matrix gives rise to computational ben- efits when making inference with GMRFs. In fact, linear algebra operations can be performed using numerical methods for sparse matrices, resulting in a considerable computational gain (see Rue and Held, 2005 for algorithms). The joint posterior distribution of and ψ is given by the product of the likelihood (4.13), of the GMRF density (4.14) and of the hyperparameter prior distribution $p(\psi)$:

$$\begin{aligned}
p(\theta, \psi \mid y) &\propto p(\psi) \times p(\theta \mid \psi) \times p(y \mid \theta, \psi) \\
&\propto p(\psi) \times p(\theta \mid \psi) \times \prod_{i=1}^{n} p\left(y_i \mid \theta_i, \psi\right) \\
&\propto p(\psi) \times |Q(\psi)|^{1/2} \exp\left(-\frac{1}{2}\theta' Q(\psi)\theta\right) \times \prod_{i=1}^{n} \exp\left(\log\left(p\left(y_i \mid \theta_i, \psi\right)\right)\right) \\
&\propto p(\psi) \times |Q(\psi)|^{1/2} \exp\left(-\frac{1}{2}\theta' Q(\psi)\theta + \sum_{i=1}^{n} \log\left(p\left(y_i \mid \theta_i, \psi\right)\right)\right)
\end{aligned}$$

5.4 Approximate Bayesian inference with INLA

The objectives of Bayesian inference are the marginal posterior distributions for each element of the parameter vector

$$p\left(\theta_i \mid y\right) = \int p\left(\theta_i, \psi \mid y\right) \mathrm{d}\psi = \int p\left(\theta_i \mid \psi, y\right) p(\psi \mid y) \mathrm{d}\psi$$

and for each element of the hyperparameter vector

$$p\left(\psi_k \mid y\right) = \int p(\psi \mid y) \mathrm{d}\psi_{-k}$$

Thus, we need to perform the following tasks: (i) compute $p(\psi \mid y)$, from which also all the relevant marginals p(k|y) can be obtained; (ii) compute $p\left(\theta_i \mid \psi, y\right)$ which is needed to compute the parameter marginal posteriors $p\left(\theta_i \mid y\right)$

The INLA approach exploits the assumptions of the model to produce a numerical approximation to the posteriors of interest based on the Laplace approximation method introduced in Section 4.7 (Tierney and Kadane, 1986). The first task (i) consists of the computation of an approximation to the joint posterior of the hyperparameters as:

$$\begin{aligned}
p(\psi \mid y) &= \frac{p(\theta, \psi \mid y)}{p(\theta \mid \psi, y)} \\
&= \frac{p(y \mid \theta, \psi) p(\theta, \psi)}{p(y)} \frac{1}{p(\theta \mid \psi, y)} \\
&= \frac{p(y \mid \theta, \psi) p(\theta \mid \psi) p(\psi)}{p(y)} \frac{1}{p(\theta \mid \psi, y)} \\
&\propto \frac{p(y \mid \theta, \psi) p(\theta \mid \psi) p(\psi)}{p(\theta \mid \psi, y)} \\
&\approx \left.\frac{p(y \mid \theta, \psi) p(\theta \mid \psi) p(\psi)}{\tilde{p}(\theta \mid \psi, y)}\right|_{\theta=\theta^* \psi} =: \tilde{p}(\psi \mid y)
\end{aligned}$$

where $\tilde{p}(\theta \mid \psi, y)$ is the Gaussian approximation – given by the Laplace method – of $p(\theta \mid \psi, y)$ and $\theta^*(\psi)$ is the mode for a given ψ the Gaussian approximation

turns out to be accurate since $p(\theta \mid \psi, y)$appears to be almost Gaussian as it is a priori dis- tributed like a GMRF, y is generally not informative and the observation distribution is usually well-behaved. The second task (ii) is slightly more complex, because in general there will be more elements in than in , and thus this computation is more expensive. A first easy possibility is to approximate the posterior conditional distributions $p(\theta \mid \psi, y)$ directly as the marginals from $\tilde{p}(\theta \mid \psi, y)$ i.e. using a Normal distribution, where the Cholesky decomposition is used for the precision matrix (Rue and Martino, 2007). While this is very fast, the approximation is generally not very good. The second possibility is to rewrite the vector of parameters as $\theta = (\theta_i, \theta_{-i})$ and use again Laplace approximation to obtain

$$\begin{aligned} p(\theta_i \mid \psi, y) &= \frac{p((\theta_i, \theta_{-i}) \mid \psi, y)}{p(\theta_{-i} \mid \theta_i, \psi, y)} \\ &= \frac{p(\theta, \psi \mid y)}{p(\psi \mid y)} \frac{1}{p(\theta_{-i} \mid \theta_i, \psi, y)} \\ &\propto \frac{p(\theta, \psi \mid y)}{p(\theta_{-i} \mid \theta_i, \psi, y)} \\ &\approx \left. \frac{p(\theta, \psi \mid y)}{\tilde{p}(\theta_{-i} \mid \theta_i, \psi, y)} \right|_{\theta_{-i} = \theta^*_{-i}(\theta_i, \psi)} =: \tilde{p}(\theta_i \mid \psi, y) \end{aligned}$$

where $\tilde{p}(\theta_{-i} \mid \theta_i, \psi, y)$ is the Laplace Gaussian approximation to $p(\theta_{-i} \mid \theta_i, \psi, y)$ and $\theta^*_{-i}(\theta_i, \psi)$ is its mode. Because the random variables $\theta_{-i} \mid \theta_i, \psi, y$ re in general reasonably Normal, the approximation provided by (4.20) typically works very well. This strategy, however, can be very expensive in computational terms as $\tilde{p}(\theta_{-i} \mid \theta_i, \psi, y)$ must be recomputed for each value of and (some modifications to the Laplace approximation in order to reduce the computational costs are described in Rue et al., 2009).

Operationally, INLA proceeds as follows: (i) first it explores the hyperparameter joint posterior distribution $\tilde{p}(\psi \mid y)$ of Eq. (4.18) in a nonparametric way, in order to detect good points $\{\psi^{(j)}\}$ for the numerical integration required in Eq. (4.22). Rue et al. (2009) propose two different exploration schemes, both requiring a reparameterization of the ψ-space – in order to deal with more

regular densities – through the following steps:

a) Locate the mode ψ^* of $\tilde{p}(\psi \mid y)$ by optimizing $\log \tilde{p}(\psi \mid y)$ with respect to (e.g., through the Newton–Raphson method).
b) Compute the negative Hessian H at the modal configuration.
c) Compute the eigen-decomposition $= V\Lambda^{1/2}V'$, with $\Sigma = H^{-1}$.
d) Define the new variable z, with standardized and mutually orthogonal components, such that:

$$\psi(z) = \psi^* + V\Lambda^{1/2}z$$

The first exploration scheme (named grid strategy) builds, using the z-parameterization, a grid of points associated with the bulk of the mass of $\tilde{p}(\psi \mid y)$. This approach has a computational cost which grows exponentially with the number of hyperparameters; therefore the advice is to adopt it when K, the dimension of ψ, is lower than 4. Otherwise, the second explo- ration scheme, named central composite design (CCD) strategy, should be used as it reduces the computational costs. With the CCD approach, the integration problem is seen as a design problem; using the mode ψ^* and the Hessian H, some relevant points in the -space are selected for performing a second-order approximation to a response variable (see Section 6.5 of Rue et al., 2009 for details). In general, the CCD strategy uses much less points, but still is able to capture the variability of the hyperparameter distribution. For this reason it is the default option in R-INLA.

5.5 Stochastic partial differential equation

approach

As described in the previous chapters, in R-INLA the linear predictor η_i is defined using the formula which includes `f()` terms for nonlinear effects of

covariates or random effects. In the same way, the Matérn GF will be included in the formula using a proper specification for `f()`. For making the connection between the linear predictorη i and the formula more explicit, it is convenient to follow Lindgren and Rue (2015) and to rewrite the linear predictor as:

$\eta_i = \sum_k h_k\left(z_i^k\right)$

where k denotes the k th term of the `formula`, z_i^k represents the covariate value for a fixed/nonlinear effect or, in the case of a random effect, the index of a second-order unit (e.g., area or point ID). The mapping function $h_k(\cdot)$ links z_i^k to the actual value of the latent field for the k th formula component. As an example consider the case where the linear predictor η_i includes a fixed effect of a covariate (named `z1` in R), a nonlinear effect (e.g., RW2) of the variable `time` (given by a sequence of time points), and a random effect with iid components indexed by the variable `index.random`. Thus the corresponding `R-INLA formula` is

```
formula <- -1 + z1 + f(time, model="rw2") + f(index.random, model="iid")
```

The default intercept is not included as we specify `-1` in the formula. With the new linear pwredictor, we have that $h_1\left(z_i^1\right)$ is equal to $z_i^1\beta$, $h_2\left(z_i^2\right)$ is the smooth effect evaluated at z_i^2 (the ith element of vector time), and $h_3\left(z_i^3\right)$ is the random effect component with index equal to z_i^3 (the ith element of `index.random`). As usual, the latent field θ is the joint vector of all the latent Gaussian variables included in the linear predictor.

The formulation only allows each observation to directly depend on a single element z_i^k from each effect $h_k(\cdot)$ and this does not cover the case when a random effect is defined as a linear combination of temporal or areal values, such as the SPDE representation. In such cases each observation y_i depends on a linear combination of the elements of and the observation distribution will be defined as:

$y_i \mid \theta, \psi \sim p\left(y_i \mid \sum_j A_{ij}\theta_j, \psi\right)$

instead of $y_i \mid \theta_i, \psi \sim p\left(y_i \mid \theta_i, \psi\right)$ as in Eq. (4.13). The term A_{ij} is the generic element of the matrix $\mathbf{A}$ referred to as the observation or projector matrix. In Sections 6.7.2 and 6.7.3, we will show how to create the $\mathbf{A}$ matrix using the helper func tion `inla.spde.matrix.A` and how to include it in the inla call through the `control.predictor` option. The $\mathbf{A}$ matrix defines a mapping between the spatial latent field (defined on the mesh) and the observations (defined in a set of locations) and this allows the SPDE models to be treated as standard indexed random effects (the mapping is done by placing appropriate $\varphi_g(s)$ values in the $\mathbf{A}$ matrix, see Eq. (6.18)). Internally, `R-INLA` creates a new linear predictor $\eta^\star$ defined as a linear combination of the original one η:

$\eta^\star = A\eta$

and in this case the likelihood is linked to the latent field through$\eta_i^\star$ instead of η_i:

$p(y \mid \theta, \psi) = \prod_{i=1}^{n} p\left(y_i \mid \eta_i^\star, \psi\right)$

5.6 The Projector Matrix

We need to construct a projection matrix $\mathbf{A}$ to project the GRF from the observations to thetriangulation vertices. The matrix $\mathbf{A}$ as the number of rows equal to the number of observations, and the number of columns equal to the number of vertices of the triangulation. Row i of $\mathbf{A}$ corresponding to an observation at location s_i ossibly has three non-zero values at the columns that correspond to the vertices of the triangle that contains the location. If s_i within the triangle, these values are equal to the barycentric coordinates. That is, they are proportional to the areas of each of the three subtriangles defined by the location s_i and the triangle's vertices, and sum to 1. If s_i s equal to a vertex of the triangle, row
i has just one non-zero value equal to 1 at the column that corresponds to the vertex. Intuitively, the value $Z(s)$ at a location that lies within one triangle is the projection of the plane formed by the triangle vertices weights at location

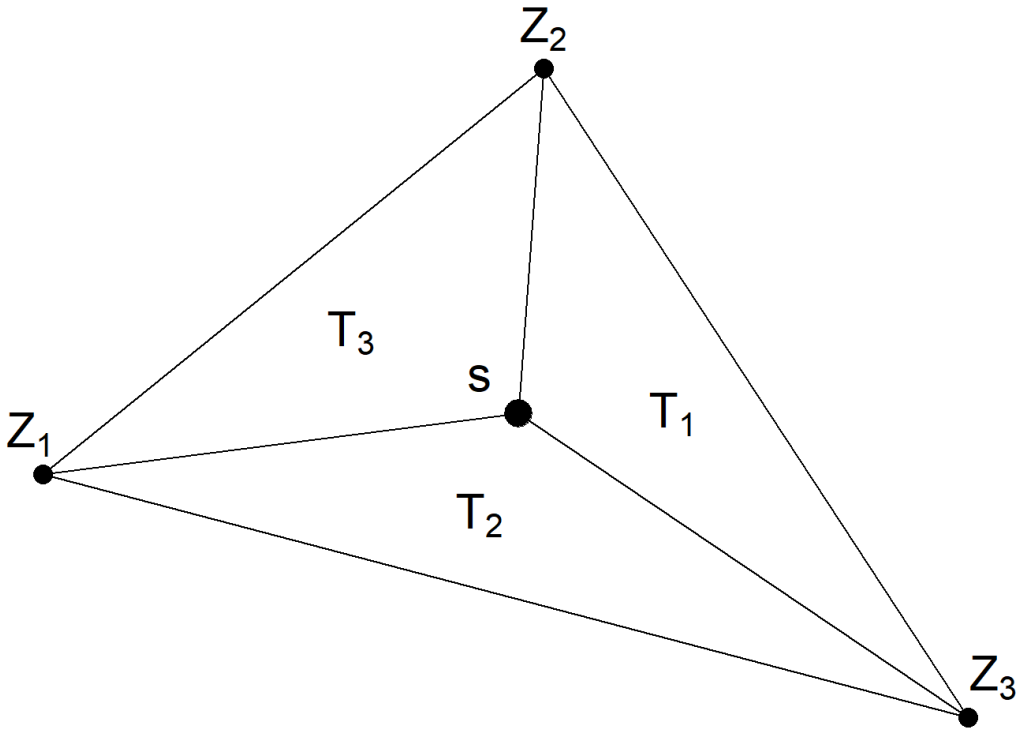
s .

An example of a projection matrix is given below. This projection matrix projects n observations to G triangulation vertices. The first row of the matrix corresponds to an observation with location that coincides with vertex number 3. The second and last rows correspond to observations with locations lying within triangles.

$$4 = \begin{bmatrix} A_{11} & A_{12} & A_{13} & \dots & A_{1G} \\ A_{21} & A_{22} & A_{23} & \dots & A_{2G} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & A_{n3} & \dots & A_{nG} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & \dots & 0 \\ A_{21} & A_{22} & 0 & \dots & A_{2G} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & A_{n3} & \dots & 0 \end{bmatrix}$$

igure 8.6 shows a location s that lies within one of the triangles of a triangulated mesh. The value of the process $Z(\cdot)$ at s is expressed as a weighted average of the values of the process at the vertices of the triangle (Z_1, Z_2 and Z_3)and with weights equal to T_1/T, T_2/T and T_3/T where Tdenotes the area of the big triangle that contains s and T_1, T_2 and T_3 are the areas of the subtriangles

$$Z(s) \approx \frac{T_1}{T} Z_1 + \frac{T_2}{T} Z_2 + \frac{T_3}{T} Z_3$$



R-INLA provides the `inla.spde.make.A()` function to easily construct a

projection matrix **A** We create the projection matrix of our example by using `inla.spde.make.A()` passing the triangulated mesh `mesh` and the coordinates `coo`.

```
A <- inla.spde.make.A(mesh = mesh, loc = coo)
```

Chapter 6

Point Referenced Data Modelling

Chapter 7

Applications

Some *significant* applications are demonstrated in this chapter.

7.1 Example one

7.2 Example two

Chapter 8

Final Words

We have finished a nice book.

Bibliography

(2020). 7.2. advantages of using docker red hat enterprise linux 7.

(2020). An api generator for r.

(2020). Get docker.

(2020). Presentazione dei file robots.txt - guida di search console.

(2020). User agent: Learn your web browsers user agent now.

Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., Chang, W., and Iannone, R. (2020). *rmarkdown: Dynamic Documents for R*. R package version 2.3.

Chang, W., Cheng, J., Allaire, J., Xie, Y., and McPherson, J. (2020). *shiny: Web Application Framework for R*. R package version 1.4.0.2.

Densmore, J. (2019). Ethics in web scraping.

Inc, F., Weststrate, M., Russell, K., and Dipert, A. (2020). *reactR: React Helpers*. R package version 0.4.3.

Meissner, P. and Ren, K. (2020). *robotstxt: A 'robots.txt' Parser and 'Webbot'/'Spider'/'Crawler' Permissions Checker*. R package version 0.7.7.

Microsoft and Weston, S. (2020). *foreach: Provides Foreach Looping Construct*. R package version 1.5.0.

Perepolkin, D. (2019). *polite: Be Nice on the Web*. R package version 0.1.1.

Trestle Technology, LLC (2018). *plumber: An API Generator for R*. R package version 0.4.6.

Vaughan, D. and Dancho, M. (2018). *furrr: Apply Mapping Functions in Parallel using Futures*. R package version 0.1.0.

was previously an Economist at the Indeed Hiring Lab, A. F. F. (2020). Indeed tech skills explorer: Today's top tech skills.

Wikipedia (2020). Web scraping — wikipedia, l'enciclopedia libera. [Online; in data 15-luglio-2020].

Wikipedia contributors (2020). Cron — Wikipedia, the free encyclopedia. [Online; accessed 15-September-2020].

Wikiversità (2020). Scheduling — wikiversità,. [Online; accesso il 15-settembre-2020].

Xie, Y. (2016). *bookdown: Authoring Books and Technical Documents with R Markdown*. Chapman and Hall/CRC, Boca Raton, Florida. ISBN 978-1138700109.

Xie, Y. (2020). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.20.