Ignorance Scores workflow: 1) Retrieve Observations from GBIF

Alejandro Ruete

20 June 2016

This first tutorial has for aim to help you downloading all the data required to calculate Ignorance Scores and Ignorance Maps [REF?], namely, species oservations for a complete reference taxonomic group (RTG, read more about RTG in the accompanig documentation) from the GBIF http://www.gbif.org/ (http://www.gbif.org/) using R and the 'rgbif' package. I will show you some ways to download data in parallel (using multiple CPU cores for setting simultaneaous queries). This tutorial is based on the great job done by rOpenSci https://ropensci.org/ (https://ropensci.org/) producing and documenting the rgbif package documentation (https://cran.r-project.org/web/packages/rgbif/index.html) and vignette (https://cran.r-project.org/web/packages/rgbif/vignettes/rgbif_vignette.html) and from Scott Chamberlain's https://github.com/sckott (https://github.com/sckott) and Katie Heineman and Mike Gahan's blog post https://sites.google.com/site/mikegahan1/rgbif (https://sites.google.com/site/mikegahan1/rgbif). Other ways to download the data are directly from the GBIF site, or via the QGBIF package https://github.com/BelgianBiodiversityPlatform/qgis-gbif-api (https://github.com/BelgianBiodiversityPlatform/qgis-gbif-api) for QuantumGIS http://www.qgis.org/ (http://www.qgis.org/).

The procedures I show you here are aimed to download raw data with minimal filters or "curations", but there are other tools, protocols and workflows the will help you getting the data fit-for-use (see links and comments at the end of this document [^1]). Depending on your questions, explorations of the bias in sampling effort using Ignorance Scores could be performed before or after such curations.

Enough with the prelude

You will need the following R packages installed for this exercises.

```
require(rgbif)
require(foreach)
require(doParallel) ## If running on Windows
require(doMC) ## If running on Linux
require(plyr)
```

Note: parallelization is easier on Linux or iOS for which this article may be also helpful https://sites.google.com/site/mikegahan1/rgbif (https://sites.google.com/site/mikegahan1/rgbif)

Obtain species occurrence data for the Reference Taxonomic Group (RTG)

First of all, you need to identify the RTG. You can find more information about how to define an RTG here (http://bdj.pensoft.net/articles.php?id=5361), but it is basically a set of species that are likely to be observed by the same observer using a particular methodology or survey protocol. For example, birds are RTG that is very easy to define, as most ornitologist would be likely to identify other birds species appart from the ones they are reporting. However, bats (*sensu lato*) are a very particular group of mammals that may not be convinient to mix in an RTG with other land mammals (not to mention with acuatic mammals!). RTG may

also be a group of unrelated species that for some reason are being looked for within the survey (e.g. fungi and lichen species that are indicators of forest interior conditions). Then, you need to get a list of all accepted species names and keys within that group. This could be more or less cumbersome depending on if the group is a well defined taxonomic group (e.g. birds, amphibians, vascular plants), or if it is a custom list of species. GBIF relies on the biodiversity data standard DarwinCore (https://en.wikipedia.org/wiki/Darwin_Core), where there are keys to identify all species at different taxonomic levels (e.g. kingdomKey, phylumKey, classKey, orderKey, familyKey, genusKey and speciesKey). We need those to query the data later on.

Keys for custom specie list

If you have a custom made species list as RTG you need to look for the speciesKey of each species in your list with these scripts below. If you are sure about the species names you could use the `name_backbone` command, wich will give you among other things the usageKey that includes all subordinate names (varieties, sinonims and subpecies). Else, the `name_suggest` command retrieves a list of up to 20 name usages that are somehow associated with the one you, ordered by relevance. But be careful, the first one in the list may not be the one you are looking for.

```
splist <- c('Cyanocitta stelleri', 'Junco hyemalis', 'Aix sponsa')
keys <- sapply(splist, function(x) name_suggest(x)$key[1], USE.NAMES=FALSE)
#[1] 7192170 6173536 2498387
keys <- sapply(splist, function(x) name_backbone(x)$usageKey, USE.NAMES=FALSE)
#[1] 2482598 2492010 2498387
```

Keys for a taxonomic group

Instead, if you set the RTG after a well defined taxonomic group, you could either manually look for the higher taxon key that identifies the RTG by looking for this group at the GBIF Backbone (http://www.gbif.org/dataset/d7dddbf4-2cf0-4f39-9b2a-bb099caae36c), or search for it using the `name_backbone` function in the rgbif package. Suppose you forgot all about the courses in Taxonomy you took and you don?t know if Amphibia was the class or the order, and in that case, what is the key for that. Then, you search for a genus or species you know belong to the group to explore different taxonomic keys:

```
name_backbone(name='Rana', kingdom='animals')
# $usageKey
# [1] 2422253
# $scientificName
# [1] "Rana Linnaeus, 1758"
# $canonicalName
# [1] "Rana"
# $rank
# [1] "GENUS"
# $status
# [1] "ACCEPTED"
# $confidence
# [1] 94
# $matchType
# [1] "EXACT"
# $kingdom
# [1] "Animalia"
# $phylum
# [1] "Chordata"
# $order
# [1] "Anura"
# $family
# [1] "Ranidae"
# $genus
# [1] "Rana"
# $kingdomKey
# [1] 1
# $phylumKey
# [1] 44
# $classKey
# [1] 131
# $orderKey
# [1] 952
# $familyKey
# [1] 6746
# $genusKey
# [1] 2422253
# $synonym
# [1] FALSE
# $class
# [1] "Amphibia"
```

Now we know that Amphibia is a class within Chordata, and its key is 131. From now on we will work with this RTG.

Get all amphibians species names in the group

Now we need to search for all valid species names in that RTG. We do this in parallel were each core looks for chunks of 1000 (my arbitrary number) unique species names at a time. We iterate until nothing new is found. In this case 17 iterations are enough. If you set it too large some queries will end up empty resulting in error when combining them. You could even do an accumulation curve for accepted names to see if your search is close to an asymptote in the number of accepted species.

```
require(foreach)
require(doParallel)
iter<-18 # Number of blocks of 1000 species names to look for
nc<-parallel::detectCores() # Number of cores to engage
key<- 131 # Higher Taxon Key to look for, here class = Amphibia

cl<-makeCluster(nc, type=ifelse(.Platform$OS.type=="windows", "PSOCK", "FORK"))
registerDoParallel(cl)
spnames<-foreach(i = 1:iter, .combine=rbind, .packages=c("rgbif")) %dopar%
    name_lookup(rank = "Species", higherTaxonKey = key,
                start = (i-1) * 1000-1, limit = 1000, return="data")[, c("speciesKey", "speci
es", "taxonomicStatus")]
                #so far we only need these three variables
stopCluster(cl)

dim(spnames)
#[1] 18000    3
```

We now have 17000 species names of which 10365 are unique species keys `length(unique(spnames$speciesKey))` , given that many are repeated, and some other are hybrids, synonyms, have recognised problems in their names or are now clasified as subpecies of some other species. There are plenty of such taxonomical issues. The safest may be to use only accepted full species names. Only 7662 are accepted species. Once again, there are many tools to clean your data set afterwards, and you may even want to work with the raw dataset as all observations do represent some sampling effort.

```
spnames<-spnames[which(spnames$taxonomicStatus=="ACCEPTED"),]

keys<-spnames$speciesKey
names(keys)<-spnames$species

## alternatively you could remove names with particular issues in its taxonomy
# wNA<-which(is.na(names(keys)))
# if(length(wNA)>0) keys<-keys[-wNA]
#
# wEM<-which(names(keys)=="spec.")
# if(length(wEM)>0) keys<-keys[-wEM]

#Species (first and last 10)
names.tmp<-names(keys)
print(names.tmp[order(names.tmp)] [1:10])
print(names.tmp[order(names.tmp)] [(length(names.tmp)-9):length(names.tmp)])
```

Search for species occurrences in a time and place, here Europe 2000-2015

Now, let's say the study is focused only in Europe, between 2000 and 2015. We need to refine the species list only including those that occur in Europe, not to serch for clog the server with queries about species that are not even found in the area required. Alternatively, you could restrict the search by countries (country =) by ISO country codes (https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2) or within polygons (geometry =) described in Well Known Text (WKT) format.

```
years<-"2000,2015" #note, for the API to work it should be in the form stated not as a range
 or vector

cl<-makeCluster(nc, type=ifelse(.Platform$OS.type=="windows", "PSOCK", "FORK"))
registerDoParallel(cl)
res2<-foreach(i = 1:length(keys), .combine=rbind, .packages="rgbif") %dopar%
          occ_search(taxonKey=keys[i], hasCoordinate = TRUE, return = 'data',
              continent = "europe", year = years, limit=1)[1]
stopCluster(cl)

# Get a vector with a refined list of keys that occur in the area of interest
wNF<-which(res2$name=="no data found, try a different search")
if(length(wNF)>0) ref.keys<-keys[-wNF]
```

NOW, search for `nobs` number of cases for each species. If you need to get more than 200.000 obs per species, you need to re-do this process using , start = 200000 or repeat by year, or country. This may be the case if you observe a particular bias in the distribution of the observations, e.g. sometimes, the obs you get are the last 2000000 that were uploaded from a particular country Note 1: that the function occ_search accepts many keys so you could pass in all 1000 names, but splitting up in chunks of 250 would allow all to be retrieved faster Note 2: the default record limit in occ_search is 20 records, so we are getting a max of 20 records back per species, but that can be changed of course

```
require(plyr)
nobs<- 200000 # maximum Number of observations to search per species Hard limit 200,000

cl<-makeCluster(nc, type=ifelse(.Platform$OS.type=="windows", "PSOCK", "FORK"))
registerDoParallel(cl)
chunksof10 <- split(ref.keys, ceiling(seq_along(ref.keys)/10)) # Note that there is no point
 in setting chunks longer than the species list
data <- llply(chunksof10, occ_search, hasCoordinate = TRUE, return = 'data',
             continent = "europe", year = years, .parallel = TRUE, limit=nobs)
stopCluster(cl)

##Collapse to a single data.frame and inspect
all <- ldply(data,ldply)
head(all); str(all)

#Map a few of the species (mapping 1000 species on a single map would be too much)
library("ggplot2")
gbifmap(all[,], mapdatabase = "world")
```

Alternatively, send a request to GBIF and wait for it in your mail, but the total maximum number of observations would be 200,000, not 200,000 per species

```
##DOWNLOAD sends a query to GBIF
occ_download('taxonKey = 2426789', 'hasCoordinate = TRUE', "continent = europe", "year = '200
0,2015'",
             user = "USER_NAME_HERE", pwd = "YOUR_PASS_HERE", email = "YOUR MAIL HERE")
```

Finally, for most of the cases you are doing all this to compare/evaluate the bias of the data available for a single focal species. Then… do this

```
require(rgbif)
### # Search for one or many species
# splist <- c('Rana temporaria')
# keys <- name_suggest(splist)$key[1]
key <- 2426805 #name_suggest(splist)$key[1] #2426805
rana.res<-occ_search(taxonKey=key, hasCoordinate = TRUE, year = yr, limit=nobs,
return='data')

##Plot dat
require(ggplot2)
gbifmap(rana.res,mapdatabase = "world")
```

[^1]:Tools and protocols for quality checks and data filtering

remove data rows with certain issue classes

```
### remove data rows with certain issue classes
library('magrittr')
res %>% occ_issues(gass84)
```

Taverna workflow at BioVel https://www.biovel.eu/application-areas/data-refinement-wf (https://www.biovel.eu/application-areas/data-refinement-wf) Biogeo: an R package for assessing and improving data quality of occurrence record data sets http://onlinelibrary.wiley.com/doi/10.1111/ecog.02118/full (http://onlinelibrary.wiley.com/doi/10.1111/ecog.02118/full)

https://cran.r-project.org/web/packages/rgbif/vignettes/issues_vignette.html (https://cran.r-project.org/web/packages/rgbif/vignettes/issues_vignette.html) http://openrefine.org/ (http://openrefine.org/) Name validation using Open refine http://www.gbif.org/resource/81222 (http://www.gbif.org/resource/81222) Report of the Task Group on GBIF Data Fitness for Use in Distribution Modelling http://www.gbif.org/resource/82612 (http://www.gbif.org/resource/82612) GBIF Position Paper on Future Directions and Recommendations for Enhancing Fitness-for-Use Across the GBIF Network http://www.gbif.org/resource/80623 (http://www.gbif.org/resource/80623) Presentation: Introduction to Fitness-For-Use http://www.gbif.org/resource/81831 (http://www.gbif.org/resource/81831)