Script esempio random forest

Michy Alice

June 16, 2016

Introduzione: caricamento pacchetto.

Il seguente script funziona con qchlorophyll versione 0.4 o superiore.

Prima di installare qchlorophyll è necessario installare i seguenti pacchetti:

1. grid
2. lazyeval
3. randomForest

```
#install.packages("/home/qchlorophyll_0.4.tar.gz", repos = NULL, type = "source")
require(qchlorophyll)
# Percorso contenente file .csv singoli e cartelle contenenti .csv annuali
# per ogni variabile
path <- getwd()
```

Caricamento dati

Carico il dataframe di riferimento. Il dataframe di riferimento è un dataframe esportato dopo l'analisi kmeans effettuata precedentemente. Il dataframe di riferimento deve contenere longitudine, latitudine, id_pixel e gruppo. Dove il gruppo non sia stato calcolato (ad esempio a causa dei troppi NA) sarà presente un valore NA.

> Nota 1: l'analisi kmeans è esportabile direttamente in .csv con la nuova funzione qcholorphyll::export_data

> Nota 2: la funzione *load_a_single_csv* è un wrapper attorno a read.csv che converte il file caricato in un dataframe di dplyr.

```
reference_df <- load_a_single_csv(file_path = "gruppi_kmeans.csv")
```

Carico tutti i .csv annuali e .csv singoli

```
# Ottengo una lista di dataframe per ogni chiamata.
dfs_bs <- load_all_csv(main_folder_path = path,
                       folder = "BS",
                       starts_with = "BS_",
                       years = c("2011", "2012", "2013", "2014"))
dfs_ws <- load_all_csv(main_folder_path = path,
                       folder = "WS",
                       starts_with = "WIND_",
                       years = c("2011", "2012", "2013", "2014"))
dfs_sst <- load_all_csv(main_folder_path = path,
                        folder = "SST",
```

```
                        starts_with = "SST_",
                        years = c("2011", "2012", "2013", "2014"))
dfs_sss <- load_all_csv(main_folder_path =  path,
                        folder = "SSS",
                        starts_with = "SSS_",
                        years = c("2011", "2012", "2013", "2014"))
dfs_sic <- load_all_csv(main_folder_path = path,
                        folder = "SIC",
                        starts_with = "SIC_",
                        years = c("2011", "2012", "2013", "2014"))
dfs_par <- load_all_csv(main_folder_path = path,
                        folder = "PAR",
                        starts_with = "PAR_",
                        years = c("2011", "2012", "2013", "2014"))

# Caricamento .csv "singoli"
bat <- load_a_single_csv(file_path = "BAT/BAT.csv")
```

Formattazione dati

Formattazione latitudine e longitudine (può richiedere un pò di tempo). Dove *reformat* = *TRUE* viene riformattata l'intera griglia di valori di longitudine e latitudine. Dove *shift* = *TRUE* viene solo effettuato uno shift della griglia.

```
dfs_bs2 <- format_lon_lat_list(df_list = dfs_bs,
                               variable = "bs",
                               reference_df = reference_df,
                               reformat = FALSE)
dfs_ws2 <- format_lon_lat_list(df_list = dfs_ws,
                               variable = "wind",
                               reference_df = reference_df,
                               reformat = TRUE)
dfs_sst2 <- format_lon_lat_list(df_list = dfs_sst,
                                variable = "sst",
                                reference_df = reference_df,
                                reformat = TRUE)
dfs_sss2 <- format_lon_lat_list(df_list = dfs_sss,
                                variable = "sss",
                                reference_df = reference_df,
                                shift = TRUE)
dfs_sic2 <- format_lon_lat_list(df_list = dfs_sic,
                                variable = "sic",
                                reference_df = reference_df,
                                reformat = TRUE)
dfs_par2 <- format_lon_lat_list(df_list = dfs_par,
                                variable = "par",
                                reference_df = reference_df,
                                reformat = FALSE)
```

Aggiunta id_pixel e gruppo clustering

```
dfs_bs3 <- add_id_pixel_and_groups(dfs_bs2, reference_dataframe = reference_df)
dfs_ws3 <- add_id_pixel_and_groups(dfs_ws2, reference_dataframe = reference_df)
dfs_sst3 <- add_id_pixel_and_groups(dfs_sst2, reference_dataframe = reference_df)
dfs_sss3 <- add_id_pixel_and_groups(dfs_sss2, reference_dataframe = reference_df)
dfs_sic3 <- add_id_pixel_and_groups(dfs_sic2, reference_dataframe = reference_df)
dfs_par3 <- add_id_pixel_and_groups(dfs_par2, reference_dataframe = reference_df)
```

Unione di tutti i dati in un unico dataframe

```
final_df <- join_data(multiple_year_data = list(dfs_bs2, dfs_ws2, dfs_sst2,
                                                dfs_sss2, dfs_sic2, dfs_par2),
                      single_year_data = list(bat, reference_df))
```

Teniamo solo i pixel dove abbiamo fatto il clustering

```
final_df <- keep_pixels_with_group(final_df, group_name = "gruppo")
```

Rimuoviamo variabili non più usate

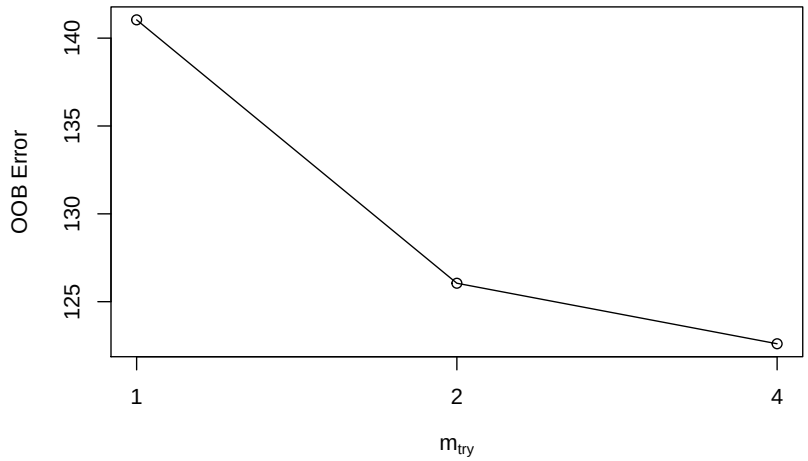
```
rm(dfs_par3,dfs_sic3,dfs_sss3,dfs_sst3,dfs_ws3,dfs_bs3,
   dfs_par2,dfs_sic2,dfs_sss2,dfs_sst2,dfs_ws2,dfs_bs2,
   dfs_par,dfs_sic,dfs_sss,dfs_sst,dfs_ws,dfs_bs,bat)
```

Modello random forest

Vediamo quale parametro potrebbe essere adatto per il parametro *mtry*. Utilizziamo 1000 alberi.

```
# Cerchiamo di ottimizzare il parametro mtry sfruttando la funzione tuneRF
# all'interno del pacchetto random forest.
data_fit <- na.omit(final_df)
fit_x <- dplyr::select(data_fit,year,wind,sst,sss,sic,par,depth,gruppo)
randomForest::tuneRF(x = fit_x, y = data_fit$bs, ntreeTry = 1000)
```

```
## mtry = 2  OOB error = 126.05
## Searching left ...
## mtry = 1     OOB error = 141.0398
## -0.1189198 0.05
## Searching right ...
## mtry = 4     OOB error = 122.6058
## 0.0273236 0.05
```




```
##   mtry OOBError
## 1    1 141.0398
## 2    2 126.0500
## 4    4 122.6058
```

Sembra che $mtry = 4$ sia la scelta ottimale in questo caso.

> Nota 1: includendo latitudine, longitudine o altro, mtry ottimale varierà.

Fit del modello

1000 alberi sono più che sufficienti (si vede dal plot sotto).

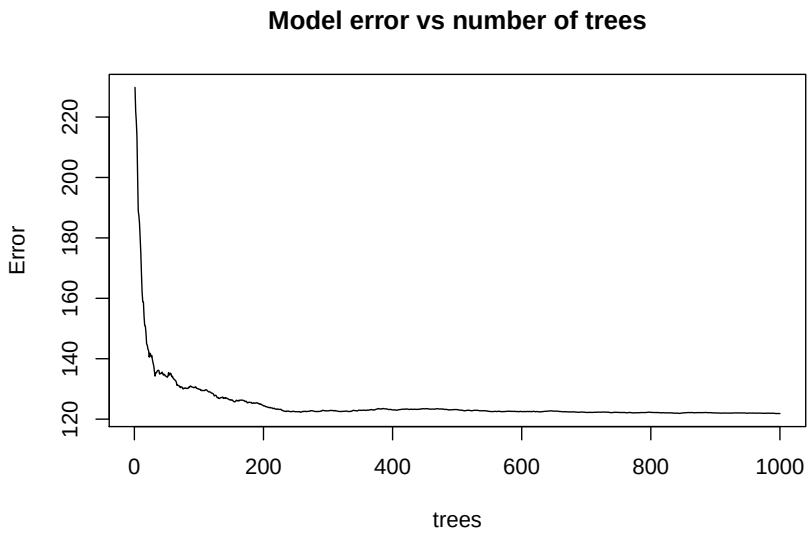
> Nota 2: Il random forest non "digerisce" bene i valori mancanti. In questo caso stiamo eliminando tutte le righe con dati mancanti utilizzando il parametro *na.action = na.omit*.

```
model <- fit_random_forest(formula = bs ~ year+wind+sst+sss+sic+par+depth+gruppo,
                           data = final_df,
                           seed = 500,
                           ntree = 1000,
                           do.trace = FALSE,
                           na.action = na.omit,
                           mtry = 4)
print(model)
```

```
##
## Call:
##  randomForest(formula = formula, data = data, ntree = ntree, importance = importance,      do.trace =
##                Type of random forest: regression
##                      Number of trees: 1000
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 121.8544
##                     % Var explained: 50.04
```

Plot errore verso numero alberi

```
plot_error_vs_trees(rf_model = model, "Model error vs number of trees")
```



Importanza variabili

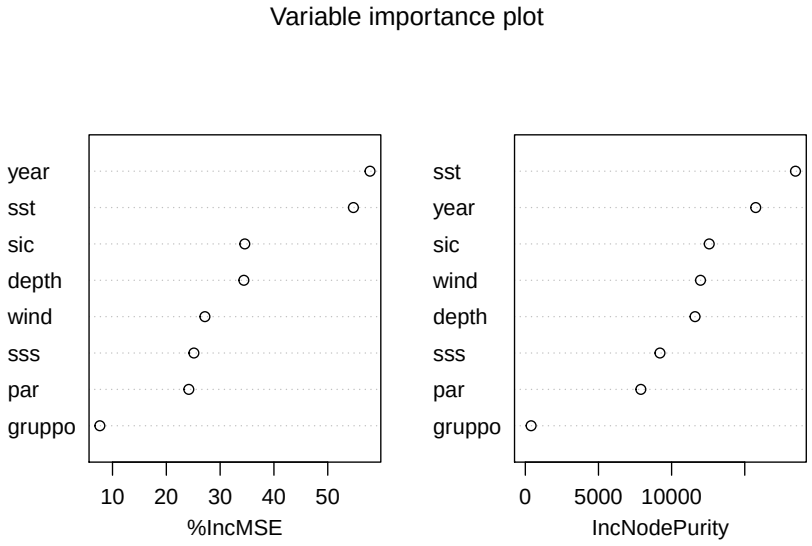
I risultati sull'importanza delle variabili possono essere ottenuti con:

```
get_variable_importance(rf_model = model)
```

```
##          %IncMSE IncNodePurity
## year   57.851901     15749.457
## wind   27.170600     11975.050
## sst    54.766517     18467.761
## sss    25.123520      9203.184
## sic    34.581255     12571.605
## par    24.176522      7896.319
## depth  34.410957     11602.385
## gruppo  7.647553       394.905
```

I risultati sopra possono essere plottati utilizzando la funzione di seguito

```
variable_importance_plot(rf_model = model)
```



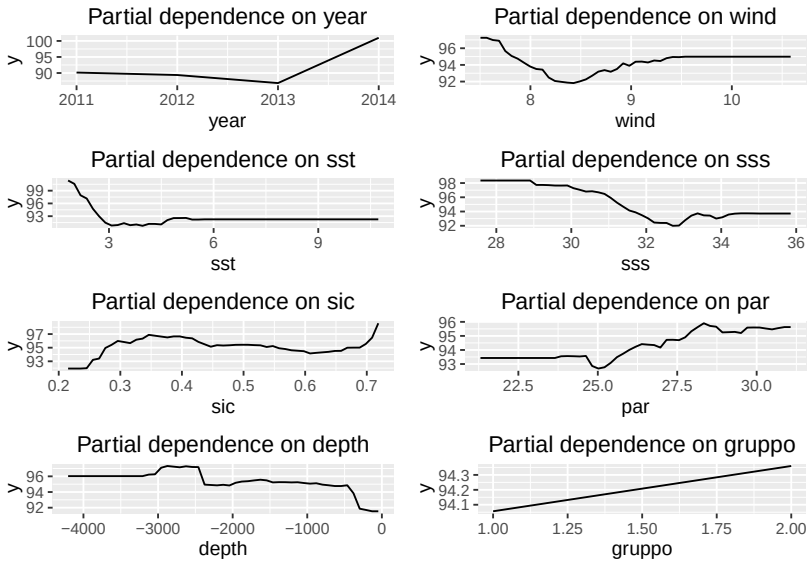
Dipendenza parziale

I dati sulla dipendenza parziale possono essere ottenuti attraverso la funzione *partial_dependence_plot* lasciando il parametro *show_plots* impostato su "FALSE" (di default è su FALSE). La funzione ritorna una lista di liste (una per ogni variabile) con i dati che verranno "plottati" dalla funzione successiva

```
pd_data <- partial_dependence_plot(model, data = final_df, show_plots = FALSE)
```

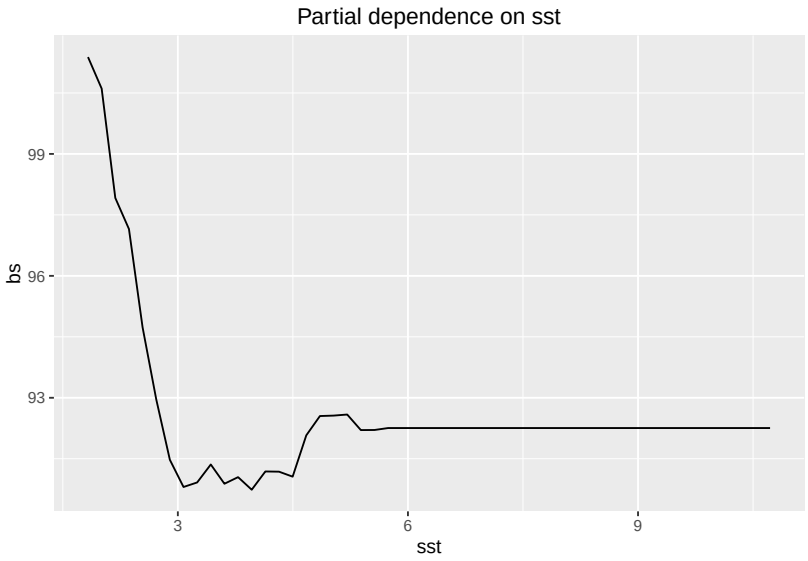
Plots della dipendenza parziale su tutte le variabili

```
partial_dependence_plot(model, data = final_df, show_plots = TRUE, cols = 2)
```



E' anche possibile effettuare il plot della dipendenza parziale su una singola variabile a scelta

```
single_partial_dependence_plot(model, final_df, "sst", ylabel = "bs")
```

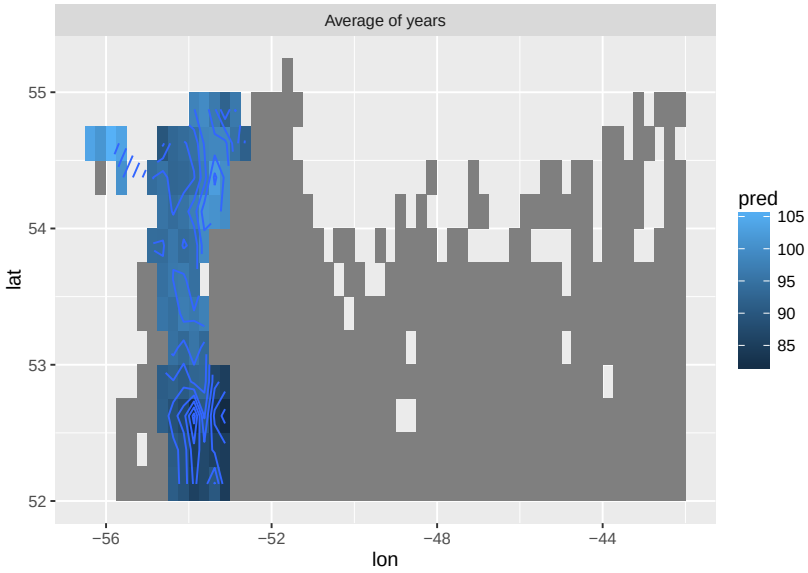


Mappa predittiva

La mappa predittiva può essere costruita utilizzando la funzione *predictive_map*. Di default, la previsione che viene utilizzata è una previsione media su tutti gli anni disponibili.

```
mp1 <- predictive_map(model, final_df)
print(mp1)
```

```
## Warning: Removed 419 rows containing non-finite values (stat_contour).
```



Se si desidera effettuare una previsione per ogni singolo anno, è possibile fissare il parametro *facet_by_year = TRUE*

```
mp2 <- predictive_map(model, final_df, facet_by_year = TRUE)
print(mp2)
```

```
## Warning: Removed 1436 rows containing non-finite values (stat_contour).
```