Script esempio random forest

Michy Alice

June 23, 2016

Introduzione: caricamento pacchetto.

Il seguente script funziona con qchlorophyll versione 1.0 o superiore.

Prima di installare qchlorophyll è necessario installare i seguenti pacchetti:

1. grid (>= 3.2.2)
2. lazyeval (>= 0.1.10)
3. randomForest (>= 4.6-12)

```
#install.packages("/home/qchlorophyll_0.4.1.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

Caricamento dataframe di riferimento

Carico il dataframe di riferimento. Il dataframe di riferimento è un dataframe esportato dopo l'analisi kmeans effettuata precedentemente oppure un dataframe che contiene solo i valori di latitudine e longitudine per ogni pixel.

Nel primo caso (che è quello che si utilizzerà di seguito), 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.

Nel secondo caso, è sufficiente che il dataframe di riferimento contenga solo i valori di latitudine e longitudine per ogni pixel.

> 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.

> Nota 3: E' necessario che i valori di longitudine e latitudine siano gli stessi per ogni file caricato (ossia il numero di pixel deve essere lo stesso, non si può caricare un file con 501 pixel distinti e un altro con 500 pixel distinti)

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

Caricamento .csv annuali

Per caricamento .csv annuali si intende il caricamento di più .csv riferiti ad una stessa variabile riferiti ad anni diversi.

Ciascuna serie di .csv deve essere contenuta in una cartella all'interno della *main_folder_path*.

Per maggiori informazioni sugli argomenti di *load_all_csv* è possibile consultare la documentazione della funzione.

```
# Per ogni chiamata ottengo una lista di dataframe

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"

E' possibile caricare .csv che si riferiscono ad un singolo anno ed andranno quindi trattati in modo differente in seguito quando verrà effettuata l'unificazione dei dati. I .csv singoli possono essere caricati utilizzando la funzione *load_a_single_csv* fornendo il percorso al file.

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

Formattazione dati

Formattazione dati "multi-anno"

Di seguito viene effettuato l'allineamento dei valori di latitudine e longitudine attraverso la funzione *format_lon_lat_list*.

La formattazione di 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. E' anche possibile fornire un valore di shift da utilizzare (vedere la documentazione della funzione).

Il nome della variabile va sempre specificato e deve essere, ovviamente, quello presente nel .csv.

```
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)
```

Formattazione .csv anno singolo

Se fosse necessario riformattare dei .csv singoli, si può procedere nel seguente modo (commentato siccome non necessario in questo esempio) attraverso le *funzioni format_lon_lat* e *shift_lon_lat*:

```
# Per riformattare l'intera griglia:
# bat2 <- format_lon_lat(dataframe = bat,
#                        reference_df = reference_df,
#                        variable = "bat")

# Per effettuare un semplice shift di 0.02 unità dei valori di
# longitudine e latitudine:
# bat2 <- shift_lon_lat(dataframe = bat,
#                       reference_df = reference_df,
#                       shift_amount = 0.02)
#
```

Aggiunta id_pixel e gruppo clustering

Aggiunta id_pixel e gruppo clustering (da utilizzare solo se il dataframe di riferimento contiene id_pixel e gruppo). La funzione aggiunge semplicemente le due variabili sopra menzionate.

```
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

Quando si uniscono tutti i dati in un unico dataframe, i .csv riferiti ad anni multipli e quelli riferiti ad anni singoli vengono gestiti separatamente dalla funzione *join_data* e quindi vanno passati attraverso due liste separate.

```
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))
```

Nel caso si voglia includere solo i pixel utilizzati nell'analisi di clustering è possibile selezionarli attraverso la funzione *keep_pixels_with_group*

> Nota: se non si esegue questa operazione rimarranno degli NA nella variabile gruppo per i pixel non utilizzati nell'analisi kmeans.

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

Rimuoviamo variabili non più usate

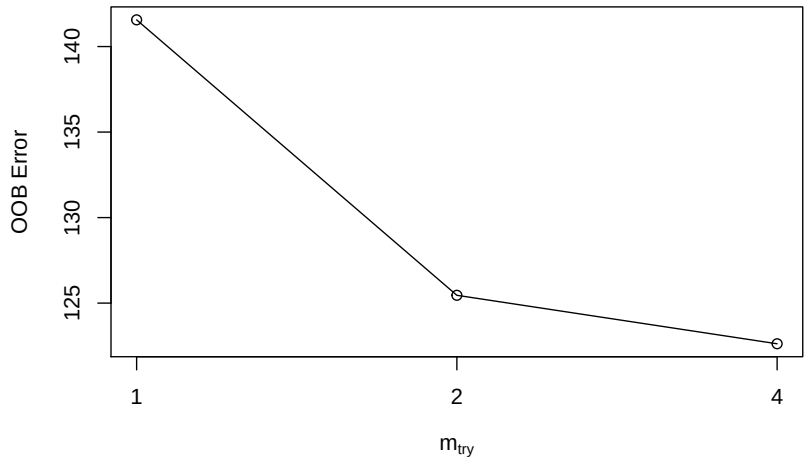
```
rm(dfs_par3,dfs_sic3,dfs_sss3,dfs_sst3,dfs_ws3,dfs_bs3,
   dfs_par2,dfs_sic2,dfs_sss2,dfs_sst2,dfs_ws2,dfs_bs2,
   dfs_par,dfs_sic,dfs_sss,dfs_sst,dfs_ws,dfs_bs,bat)
```

Modello random forest

Vediamo quale valore potrebbe essere adatto per il parametro *mtry*. Utilizziamo, ad esempio, 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 = 125.4569
## Searching left ...
## mtry = 1     OOB error = 141.5605
## -0.1283593 0.05
## Searching right ...
## mtry = 4     OOB error = 122.6236
## 0.02258378 0.05
```




```
##   mtry OOBError
## 1    1 141.5605
## 2    2 125.4569
## 4    4 122.6236
```

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, in realtà già 300 alberi sembrano sufficienti).

> 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*. E' possibile sostituire i dati mancanti con media e moda utilizzando la funzione na.roughfix al posto di na.omit.

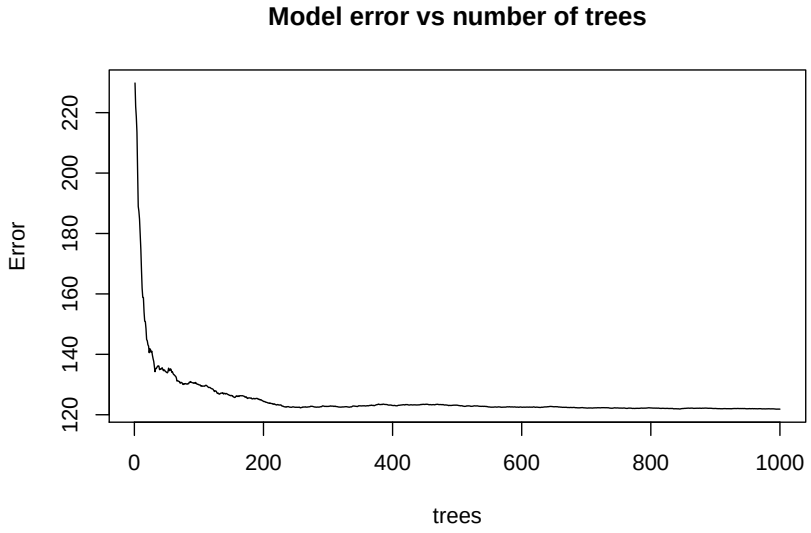
```
model <- fit_random_forest(formula = bs ~ year+wind+sst+sss+sic+par+depth+gruppo,
                           data = final_df,
                           seed = 500,
                           ntree = 1000,
                           do.trace = FALSE,
                           na.action = na.omit,
                           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, si nota chiaramente che dopo circa 300 alberi ogni albero aggiuntivo porta ad una diminuzione dell'errore quasi nulla e quindi di fatto è quasi superfluo. (Suggerisco di effettuare diverse prove).

Nota: questa funzione non restituisce un oggetto di tipo ggplot.

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



Importanza variabili

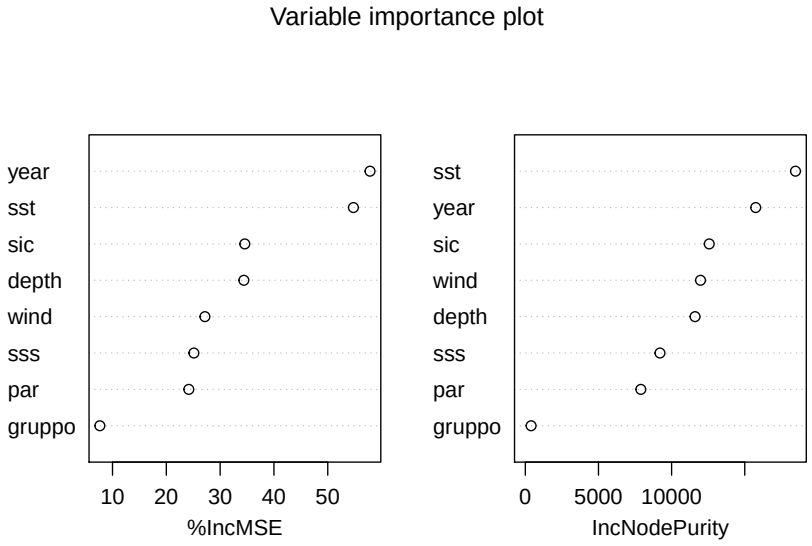
I dati calcolati sull'importanza delle variabili possono essere ottenuti con la seguente funzione:

```
var_imp_data <- get_variable_importance(rf_model = model)
```

Nota: i dati sull'importanza delle variabili sono ora raccolti in var_imp_data e possono essere utilizzati per altri plot/elaborazioni

I dati ottenuti sopra possono essere plottati per una rapida visualizzazione utilizzando la funzione *variable_importance_plot* che mostra le due statistiche di riferimento:

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



Dipendenza parziale

I dati sulla dipendenza parziale possono essere ottenuti attraverso due funzioni:

1. *partial_dependence_plot*. Questa funzione calcola i risultati sulla dipendenza parziale per tutte le variabili. Di default ritorna una lista con tanti elementi quante sono le variabili indipendenti inserite nel modello e dove ogni elemento della lista contiene i valori di ascissa e ordinata del plot della dipendenza parziale.
2. *single_partial_dependence_plot*. Questa funzione calcola i risultati sulla dipendenza parziale per una singola variabile. Di default ritorna una lista con i valori di ascissa e ordinata per il plot. Se l'argomento *return_plots = TRUE* viene ritornato un oggetto di classe *ggplot* che può essere manipolato a piacere utilizzando le varie funzioni di *ggplot2*.

partial_dependence_plot

Ottengo i dati della dipendenza parziale per tutte le variabili:

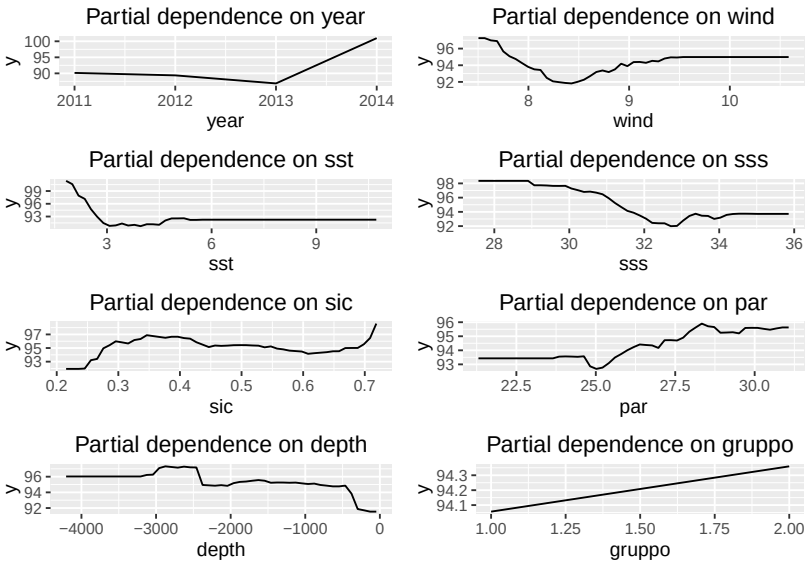
```
pd_data <- partial_dependence_plot(model, data = final_df, return_plot = FALSE)
print(names(pd_data))
```

```
## [1] "year"   "sst"    "sic"    "depth"  "wind"   "sss"    "par"    "gruppo"
```

pd_data è una lista contenente, per ogni variabile, i valori di ascissa e ordinata che servono per effettuare il plot della dipendenza parziale.

Se volessi effettuare un plot della dipendenza parziale su tutte le variabili per riferimento, è possibile fissare l'argomento *return_plots = TRUE* assieme al numero di colonne da utilizzare per mostrare i plot.

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



Nota: *partial_dependence_plot* non ritorna il plot aggregato in nessun caso. Per ottenere i plot utilizzare la funzione *single_partial_dependence_plot*

single_partial_dependence_plot

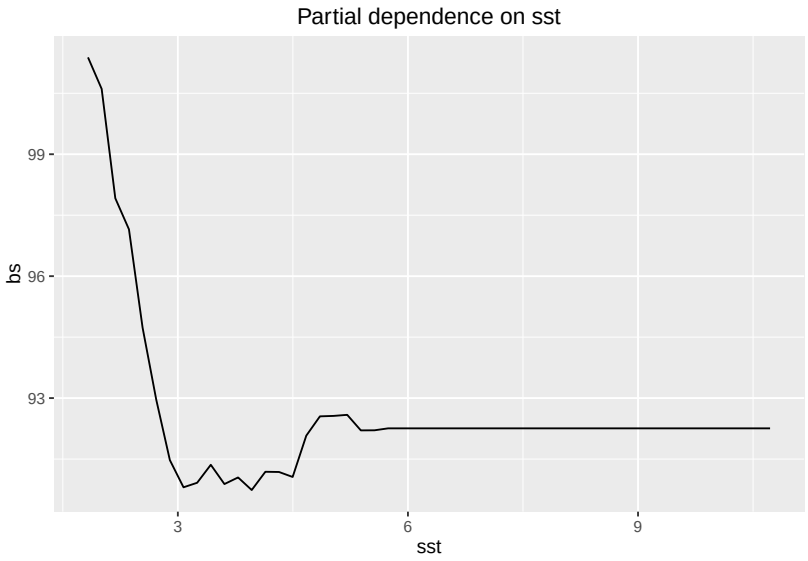
Ottengo dati dipendenza parziale per la variabile *sst*

```
pd_data_sst <- single_partial_dependence_plot(model, final_df,
                                              "sst",
                                              ylabel = "bs",
                                              return_plot = FALSE)
print(names(pd_data_sst))
```

```
## [1] "x" "y"
```

Ottengo il plot della dipendenza parziale per la variabile *sst*

```
pd_sst_plot <- single_partial_dependence_plot(model,
                                              final_df,
                                              "sst",
                                              ylabel = "bs",
                                              return_plot = TRUE)
print(pd_sst_plot)
```



Mappa predittiva

Mappa predittiva media

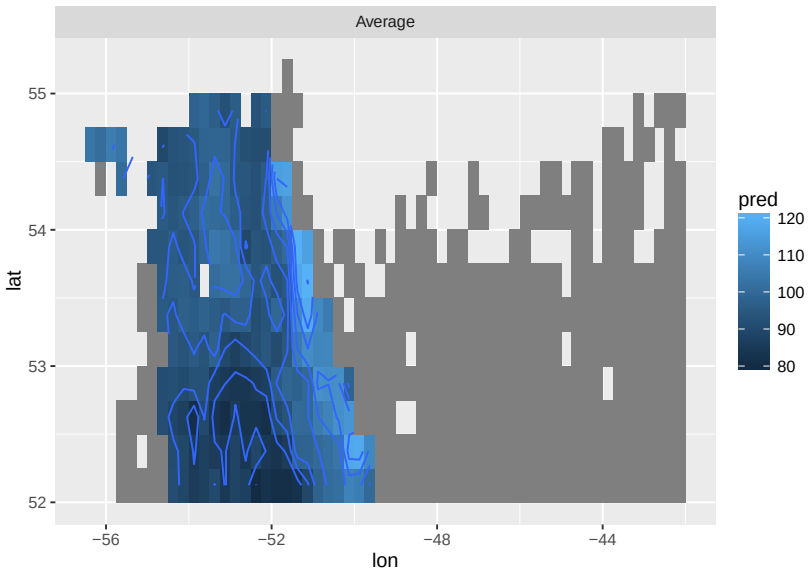
La mappa predittiva può essere costruita utilizzando la funzione *predictive_map*. Di default, la previsione che viene utilizzata è una previsione media su tutti gli anni disponibili.

> Nota 1: questa mappa è più ampia di quelle riferite al singolo anno (mappa categorizzata) siccome aggrega i valori di più anni e quindi abbiamo dei dati anche per i pixel che magari presentano dati mancanti in alcuni anni.

> Nota 2: le previsioni plottate sulla mappa sono effettuate direttamente sui dati utilizzati per il training (fit) del modello ossia, la matrice X delle variabili indipendenti è esattamente la matrice utilizzata in fit_random_forest.

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

```
## Warning: Removed 307 rows containing non-finite values (stat_contour).
```

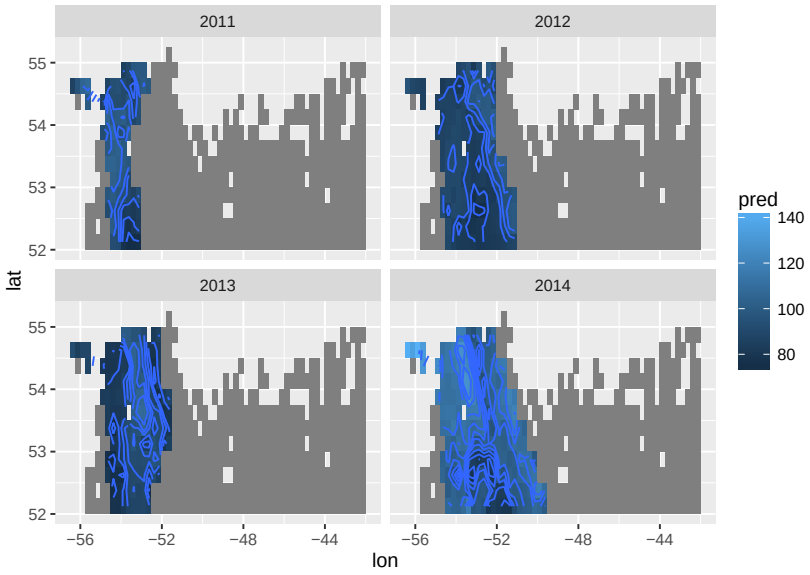


Mappa predittiva categorizzata

Se si desidera categorizzare la mappa predittiva, ad esempio per anno, è possibile utilizzare la funzione *predictive_map* e passare gli argomenti *facet_plot = TRUE* e *facet_by = "year"* che, in questo caso, dice alla funzione di utilizzare la variabile anno per categorizzare i grafici delle previsioni.

```
mp2 <- predictive_map(model, final_df, facet_plot = TRUE, facet_by = "year")
print(mp2)
```

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



Previsione valori della variabile dipendente

Utilizzando la funzione *predict* è possibile effettuare previsioni utilizzando il modello.

```r
data_fit <- na.omit(final_df)

# Matrice delle variabili da utilizzare per la previsione
# si può usare anche subset() per generarla.
#
# Nota: in x devono esserci ovviamente le variabili presenti nel modello
#
x <- dplyr::select(data_fit, year, wind, sst, sss, sic, par, depth, gruppo)
# Previsione valori di bloomstart
predicted_values <- predict(model, x)
# Valori previsti
head(predicted_values)
```

```
##         1         2         3         4         5         6
##  88.21467  97.72187  98.15160  98.09707  85.95333 104.98213
```
