Species Distribution Modelling for Marine Species

Jan Laurens Geffert, PhD - OceanOS Earth Limited

April 04, 2025

Species Distribution Models (SDMs) are essential tools for predicting where species occur, a critical need given the Wallacean shortfall - the pervasive lack of comprehensive distribution data - and the inherent imperfections in biodiversity information, especially in marine systems. Reliable predictions are necessary for effective conservation planning, yet marine environments pose unique challenges due to their vastness, dynamic nature, and less-sampled habitats.

This pipeline represents an open-source, best-practices, end-to-end workflow that integrates data from Copernicus Marine, WoRMS, and GBIF. By automating the process of downloading, cleaning, and integrating data, it significantly improves the FAIR-ness (Findability, Accessibility, Interoperability, and Reusability) of both the environmental and biological datasets as well as the resulting SDM outputs. Such a workflow ensures that data are not only available for current research but can be easily reused and built upon in the future.

A key aspect of the pipeline is the use of WoRMS data, which is crucial to filter out terrestrial taxa that may inadvertently be recorded in marine areas. This step guarantees that only true marine taxa are included, thus maintaining the ecological relevance of the analysis.

To address sampling bias - a common issue in presence-only data - the pipeline employs the target group background approach. This method corrects for bias by sampling background (pseudo-absence) points from a broader dataset of similar organisms that share the same sampling bias as the target species. In doing so, it helps ensure that the contrast between where species are observed and the overall environmental background is not unduly influenced by uneven sampling effort.

We welcome any feedback, criticism, contributions, or suggestions for improvement on this workflow. Your insights can help refine and enhance this tool, ensuring that it continues to support robust, transparent, and reproducible marine species distribution modeling.

In the first step, the Python code is designed to download high-resolution marine environmental datasets from the Copernicus Marine Service. It reads a CSV file that contains variable keys and associated dataset IDs for monthly data, then iterates through these IDs to download each dataset using a custom function from the copernicusmarine module. This automated retrieval ensures that the model is built on the comprehensive and accurate environmental information, which is critical since marine ecosystems are dynamic and sensitive to environmental changes.

```python
# Import modules
import os
import numpy as np
import xarray as xr
import pandas as pd
import copernicusmarine as cm
import scipy
from datetime import datetime, timedelta
from dateutil.relativedelta import relativedelta
import warnings

# For login information, see:
# https://help.marine.copernicus.eu/en/articles/7949409-copernicus-marine-toolbox-introduction
#
# cm.login()
# cm.login(
#   username = os.getenv("COPERNICUS_USERNAME"),
#   password = os.getenv("COPERNICUS_PASSWORD")
# )

dataset_lookup = pd.read_csv('copernicusmarine_variablekey.csv')
dataset_ids = (dataset_lookup
  .loc[dataset_lookup['timescale'] == 'monthly', 'dataset_id']
  .tolist())

def cm_download_dataset(dataset_id):
  print('downloading dataset_id: ' + dataset_id)
  cm_nws = cm.get(
    dataset_id = dataset_id,
    output_directory = 'outputs/copernicus/' + dataset_id)
  return cm_nws

x = [cm_download_dataset(dataset_id) for dataset_id in dataset_ids]
```

Next, the workflow shifts to R where the "worms_download" chunk retrieves taxonomic records from the World Register of Marine Species (WoRMS) via a REST API. By batching API requests and handling potential errors, this part of the code consolidates and writes high-quality taxonomic information to disk. Accurate taxonomic data are crucial because they help ensure that the species occurrence records are correctly identified and matched, thereby improving the reliability of the subsequent analyses.

```r
library(tidyverse)
library(curl)
library(jsonlite)

# Base URL for the request
# WORMS API with fuzzy matching on
base_url <- "https://www.marinespecies.org/rest/AphiaRecordsByName/%25?like=true&marine_only=false&offs

# Function to reformat System.time() to filename time tag
ts_lodash <- function() {
  Sys.time() %>%
    str_replace_all("[^\\w]*", "#") %>%
    str_replace_all("##", "_") %>%
    str_replace_all("#", "")
}

# Function to retrieve data
fetch_results <- function(offset) {
  # Generate request
  request_url <- base_url %>%
    paste0(offset) %>%
    URLencode(reserved = TRUE)
  # Fetch data
  response <- curl_fetch_memory(request_url)
  # Error catching
  if (response$status_code != 200) {
    message(str_glue("Response status code: {response$status_code}"))
    Sys.sleep(120)
    response <- curl_fetch_memory(request_url)
    stopifnot(response$status_code == 200)
  }
  content <- rawToChar(response$content)
  # Convert JSON to data.frame
  data <- fromJSON(content, flatten = TRUE)
  return(data)
}

# Initialize variables
all_results <- tibble() # To store all results
offset <- 1             # Initial offset
batch_size <- 50        # Maximum batch size per request

# Fetch data in batches until no more results
message(str_glue("Commencing download at {Sys.time()}"))
repeat {
  message(str_glue("Downloading batch {(offset - 1) / batch_size + 1}"))
  # Fetch current batch
  current_batch <- fetch_results(offset)
  # If current batch has no results, stop
  if (nrow(current_batch) == 0) {
    break
  }
  # Add current batch to all results
  all_results <- bind_rows(all_results, current_batch)
  # Update offset to fetch next batch
  offset <- offset + batch_size
}
message(str_glue("Completed download at {Sys.time()}"))

# Store the results
write_rds(
  x = all_results,
  file = str_glue("outputs/worms_taxonomy/dl_offset{offset}_{ts_lodash()}.rds"))
write_csv(
  x = all_results,
  file = str_glue("outputs/worms_taxonomy/download_{ts_lodash()}.csv"))

# Post-processing -------------------------------------------------------------

raw_results <- read_rds("outputs/worms_taxonomy/worms_full_download.rds")

# Check status
raw_results %>%
  pluck("status") %>%
  table()

# How many marine?
raw_results %>%
  pluck("isMarine") %>%
  table()

# Check distribution of first letter
raw_results %>%
  pluck("scientificname") %>%
  str_extract("^\\w") %>%
  table()

# Check a taxon with lowercase name
raw_results %>%
  filter(str_extract(scientificname, "^\\w") == "u") %>%
  pluck("url")

all_results <- raw_results %>%
  filter(isMarine != 0 | is.na(isMarine)) %>%
  filter(isExtinct != 1 | is.na(isExtinct)) %>%
  filter(status %in% c(
    "accepted",
    "alternative representation",
    NA_character_)) %>%
  filter(kingdom %in% c(
    "Animalia",
    "Plantae",
    "Protozoa") | is.na(kingdom)) %>%
  # Take out some records with data quality issues
  filter(AphiaID == valid_AphiaID) %>%
  # For multiple names per valid_AphiaID, only retain the first one.
  group_by(valid_AphiaID) %>%
  arrange(status) %>%
  summarize_all(.funs = first) %>%
  # Exclude higher ranks
  filter(rank %in% c(
    "Species",
    "Subspecies",
    "Variety",
    "Subvariety",
    "Forma",
    "Tribe") | is.na(rank))

# Write to disk taxa of interest for GBIF download
all_results %>%
  write_rds(file = "outputs/worms_taxonomy/taxon_shortlist.rds")
```

Armed with our taxonomic marine species list we turn to species occurrence data sourced from the Global Biodiversity Information Facility (GBIF). The code performs backbone matching to reconcile species names and then applies spatial, temporal, and quality filters during the download process. It breaks the download into manageable chunks to avoid maxing out taxon list query parameters, overwhelming the server, or creating unmanageable file sizes. Various quality filters are applied to the occurrence records to exclude uncertain, erroneous, or ambiguous records. The results are stored in local parquet files which can be loaded lazily so that unneccessary RAM spikes are avoided even for large datasets.

Please don't forget that you have to cite the GBIF download appropriately when using the resulting data! You can find the relevant information via `cite(gbif_download_list[[1]])`

```r
library(rgbif)
library(sf)
library(wk)

# Backbone matching ------------------------------------------------------------
taxa_worms <- read_rds("outputs/worms_taxonomy/taxon_shortlist.rds")

# Find ids for a long species list as outlined at:
# https://docs.ropensci.org/rgbif/articles/downloading_a_long_species_list.html
taxa_search <- mutate(
  .data = taxa_worms,
  name = str_c(scientificname, authority, sep = " "))
matches <- tibble()
n_chunks <- ceiling(nrow(taxa_search) / 1000)
i <- 1

# Iterate over chunks
# Doing it like this because it often falls over for longer lists
message(str_glue("Commencing download at {Sys.time()}"))
while (i <= n_chunks) {
  message(str_glue("Downloading chunk {i}"))
  offset <- i * 1000 - 999
  matches_new <- taxa_search %>%
    filter(row_number() %in% c(offset:(offset + 999))) %>%
    name_backbone_checklist(
      strict = TRUE,
      verbose = TRUE)
  matches <- matches_new %>%
    mutate(verbatim_index = verbatim_index + offset) %>%
    bind_rows(matches, .)
  # Increment chunk
  i <- i + 1
}
message(str_glue("Completed download at {Sys.time()}"))
taxa_gbif <- matches
write_rds(taxa_gbif, file = 'outputs/gbif/taxonmatch.rds')

# Quality control of matches ---------------------------------------------------
taxa_gbif <- read_rds(file = 'outputs/gbif/taxonmatch.rds')

# How many matches per species?
taxa_gbif %>%
  pluck("verbatim_index") %>%
  table() %>%
  sort(decreasing = TRUE) %>%
  head(100)
# # Check taxon with most matches
taxa_gbif %>%
  filter(verbatim_index == 228934) %>%
  View()
# Check distribution of confidence scores
taxa_gbif %>%
  # Flooring confidence level
  mutate(confidence = pmax(confidence, 0)) %>%
  pluck("confidence") %>%
  hist(breaks = 10)

taxa_gbif %>%
  #sample_n(1000) %>%
  group_by(verbatim_index) %>%
  nest(.key = "matches") %>%
  ungroup() %>%
  mutate(n = map_dbl(matches, nrow)) %>%
  arrange(desc(n))

taxa_gbif %>%
  filter(confidence == 100
         & matchType == "EXACT"
         & rank == "SPECIES"
         & status == "ACCEPTED"
         & synonym == FALSE) %>%
  pluck("verbatim_index") %>%
  table() %>%
  sort(decreasing = TRUE) %>%
  head(100)

taxa_gbif_filtered <- taxa_gbif %>%
  # Drop matches to higher rank and no match
  filter(!matchType %in% c("HIGHERRANK", "NONE")) %>%
  # Lower confidence score for synonyms as tie break mechanism
  mutate(confidence = case_when(
    status %in% c("SYNONYM", "DOUBTFUL") ~ confidence - 1,
    TRUE ~ confidence)) %>%
  # Keep only highest confidence match for each search string
  slice_max(
    by = verbatim_index,
    order_by = confidence,
    with_ties = TRUE)

# Check for remaining duplicate rows
taxa_gbif_filtered %>%
  pluck("verbatim_index") %>%
  table() %>%
  sort(decreasing = TRUE) %>%
  head(100)

# Occurrence download  ---------------------------------------------------------

# Restrict results
# Create search area polygon
search_area <- tibble::tibble(
  lon = c(
    13.0552250275741,
    13.0552250275741,
    -19.9444443116317,
    -19.9444443116317,
    13.0552250275741),
  lat = c(
    65.0345862230515,
    40.0333344617629,
    40.0333344617629,
    65.0345862230515,
    65.0345862230515)) %>%
  as.matrix() %>%
  list() %>%
  st_polygon() %>%
  # Set the polygon into an sf object with a CRS (coordinate reference system)
  st_sfc(crs = 4326)  # WGS84 = EPSG:4326

search_area <- search_area %>%
  st_as_text() %>%
  wk::wkt() %>%
  wk::wk_orient(direction = wk::wk_counterclockwise())

# Get taxonKeys for marine species
taxa_gbif_keys <- taxa_gbif_filtered %>%
  pluck("usageKey") %>%
  unique()

download_gbif <- function(taxon_keys, return_tibble, ...) {
  gbif_download <- occ_download(
    pred("occurrenceStatus","PRESENT"),
    # restricting to relevant species
    pred_in("taxonKey", taxon_keys),
    # specifying a polygon in wkt format for spatially restricting search.
    pred_within(search_area),
    # restricting search to commercially licenced records.
    # accepted values: CC0_1_0, CC_BY_4_0, CC_BY_NC_4_0, UNSPECIFIED, UNSUPPORTED
    pred_in("license", c(
      "CC0_1_0",
      "CC_BY_4_0",
      "UNSPECIFIED",
      "UNSUPPORTED")),
    # excluding records not appropriate for biodiversity assessments.
    pred_not(pred_in("basisOfRecord", c(
      "FOSSIL_SPECIMEN",
      "LIVING_SPECIMEN",
      "MATERIAL_SAMPLE",
      "MATERIAL_CITATION",
      "PRESERVED_SPECIMEN"))),
    # excluding records with know issues.
    pred("hasGeospatialIssue", FALSE),
    pred("hasCoordinate", TRUE),
    # excluding low spatial accuracy records.
    pred_or(
      pred_lt("coordinateUncertaintyInMeters", 3500),
      pred_isnull("coordinateUncertaintyInMeters")),
    # excluding records close to country centroid
    pred_gte("distanceFromCentroidInMeters", "2000"),
    format = "SIMPLE_PARQUET")
    #format = "SIMPLE_CSV")
  # Handle response
  occ_download_wait(gbif_download)
  if (return_tibble) {
    response <- gbif_download %>%
      occ_download_get() %>%
      occ_download_import()
  } else {
    download_info <- occ_download_get(gbif_download)
    response <- tibble(
      chunk = i + 1,
      info = list(download_info),
      status = 'done')
  }
  return(response)
}

batch_download <- function(
    x,
    f,
    batch_size,
    start_batch = 1,
    write_to_disk = FALSE,
    ...) {
  response <- tibble()
  i <- start_batch - 1
  n_chunks <- ceiling(length(x) / batch_size)
  # Iterate over chunks
  # Doing it like this because it often falls over for longer lists
  message(str_glue("Commencing batch process at {Sys.time()}"))
  while (i < n_chunks) {
    message(str_glue("Downloading chunk {i + 1}"))
    offset <- (i * batch_size)
    message(str_glue("Offset is {offset}"))
    batch <- x[(offset + 1):min((offset + batch_size), length(x))]
    message(str_glue("Batch is {length(batch)} rows long"))
    response_new <- f(batch, ...)
    # Handle response
    response <- bind_rows(response, response_new)
    # Increment chunk
    i <- i + 1
    message('')
  }
  message(str_glue("Completed download at {Sys.time()}"))
  return(response)
}
```

```r
# Execute batch download of all relevant GBIF records
set.seed(42)
gbif_download_list <- batch_download(
  x = sample(taxa_gbif_keys),
  f = download_gbif,
  batch_size = 20000,
  return_tibble = FALSE
)
```

Here, we bring together the cleaned occurrence data and the environmental variables. We begin by reading in environmental data from multiple netCDF files and processes them using spatial tools (such as stars, terra, and sf) to ensure that they are correctly aligned with the occurrence data. The code then cleans and standardizes date information, splits occurrence data into target species and background (non-target) groups, and extracts the relevant environmental values at the occurrence locations by matching them by year and month. This matching ensures that the models account for temporal variability in the marine environment, which is particularly important in ecosystems that experience rapid environmental shifts and for species that may rapidly migrate to stay within the most favourable environmental conditions.

```r
library(fs)
library(sf)
library(arrow)
library(star)
library(terra)
library(tidyterra)
library(tidysdm)

# Loading Species Data -------------------------------------------------------

# Find and index parquet files
files <- fs::dir_ls(
  path = "outputs/gbif/parquet/",
  recurse = TRUE,
  type = "file")
# Filter out files with size 0 bytes
valid_files <- files[file.info(files)$size > 0]
# Open the dataset from multiple folders (all must have the same schema)
ds <- open_dataset(valid_files, format = "parquet")

clean_dates <- function(dates) {
  dates <- dates %>%
    # Remove date ranges (keep the first date)
    str_remove("\\/.*") %>%
    # Remove time info
    str_remove("T.*")
  case_when(
    # Already YYYY-MM-DD
    str_detect(dates, "^\\d{4}-\\d{2}-\\d{2}$") ~ dates,
    # Append "-01" for YYYY-MM
    str_detect(dates, "^\\d{4}-\\d{2}$") ~ paste0(dates, "-01"),
    # Append "-01-01" for YYYY
    str_detect(dates, "^\\d{4}$") ~ paste0(dates, "-01-01"),
    # Invalid formats become NA
    TRUE ~ NA_character_) %>%
    as.Date()
}

# Collect occurrence data
df_occs <- ds %>%
  filter(phylum == "Chordata") %>%
  filter(!class %in% c('Aves', 'Insecta')) %>%
  filter(taxonrank %in% c("SPECIES", "SUBSPECIES", "VARIETY")) %>%
  filter(basisofrecord %in% c(
    "HUMAN_OBSERVATION",
    "MACHINE_OBSERVATION",
    "OBSERVATION",
    "OCCURRENCE")) %>%
  select(
    taxonkey,
    eventdate,
    decimallatitude, decimallongitude,
    occurrencestatus, individualcount) %>%
  collect() %>%
  transmute(
    taxonkey = taxonkey,
    lon = decimallongitude,
    lat = decimallatitude,
    # Occurrence record context
    occurrencestatus = as.factor(occurrencestatus),
    individualcount = individualcount,
    eventdate = clean_dates(eventdate)
    )

# UK invasive species
target_species <- c(
  # "Sargassum muticum (Yendo) Fensholt" = 3197314L,
  # "Crepidula fornicata (Linnaeus, 1758)" = 5192789L,
  # "Styela clava Herdman, 1881" = 2331942L,
  # "Eriocheir sinensis H.Milne Edwards, 1853" = 2225776L
  "Magallana gigas (Thunberg, 1793)" = 7820753L)

# Records for target species only
df_occs_target <- df_occs %>%
  filter(taxonkey %in% target_species) %>%
  filter(eventdate >= '1993-01-01') %>%
  filter(occurrencestatus == "PRESENT") %>%
  mutate(yearmonth = format(eventdate, "%Y%m")) %>%
  select(-eventdate, -occurrencestatus, -individualcount) %>%
  st_as_sf(coords = c('lon', 'lat'), crs = 4326) %>%
  distinct()

# Records for all other vertebrates
df_occs_background <- df_occs %>%
  filter(!taxonkey %in% target_species) %>%
  filter(eventdate >= '1993-01-01') %>%
  filter(occurrencestatus == "PRESENT") %>%
  mutate(yearmonth = format(eventdate, "%Y%m")) %>%
  select(-eventdate, -occurrencestatus, -individualcount) %>%
  st_as_sf(coords = c('lon', 'lat'), crs = 4326) %>%
  distinct()

# Loading Environmental Data ---------------------------------------------------

# List all netCDF files in the specified directory
all_nc_files <- dir_ls(
  path = "outputs/copernicus/raw/",
  recurse = TRUE,
  type = "file",
  glob = "*.nc")

# Helper function to deal with depth dimension
nc_read_drop_depth <- function(filepaths) {
  results <- vector("list", length(filepaths))
  for (i in seq_along(filepaths)) {
    f <- filepaths[i]
  # Read file and drop depth dim
    x <- read_stars(f)
    dims <- names(st_dimensions(x))
    if ("depth" %in% dims) {
      x <- x[ , , , 1, drop = TRUE]
    } else {
      x <- x[ , , , drop = TRUE]
    }
    results[[i]] <- x
  }
  # Stack to output
  combined <- do.call(c, c(results, list(along = NA)))
  return(combined)
}

# Function to read the netCDF file corresponding to a given year-month (ym)
read_yearmonth <- function(yearmonth, nc_files) {
  nc_files_subset <- nc_files %>%
    str_subset(pattern = str_c(yearmonth, ".nc$")) %>%
    sort()
  if (length(nc_files_subset) == 0) {
    warning("No environmental file found for yearmonth = ", yearmonth)
    return(NULL)
  }
  if (length(nc_files_subset) != 19) {
    stop("Something is wrong with env_data, not the expected 19 layers!")
  }
  # Read the netCDF file as a stars object
  env_data <- nc_files_subset[1:18] %>%
    nc_read_drop_depth() %>%
    #units::drop_units() %>%
    set_names(., names(.) %>% str_extract("(?<=NWS_).+(?=_mm\\d{6}\\.nc)"))
  return(env_data)
}
```

```r
# For each unique year-month, extract the environmental values at occurrence points
# First for target occurrences
env_target <- df_occs_target %>%
  group_by(yearmonth) %>%
  group_split() %>%
  map(function(occ_points) {
    yearmonth <- unique(occ_points$yearmonth)
    env_data <- read_yearmonth(yearmonth, all_nc_files)
    if (is.null(env_data)) return(NULL)
    # Join environmental data with occurrence points
    env_values <- stars::st_extract(
      x = env_data,
      at = st_coordinates(occ_points))
    occ_data <- bind_cols(occ_points, env_values)
    return(occ_data)
  })
df_env_target <- do.call(rbind, env_target)

# Second for background group occurrences
env_background <- df_occs_target %>%
  group_by(yearmonth) %>%
  group_split() %>%
  map(function(occ_points) {
    yearmonth <- unique(occ_points$yearmonth)
    env_data <- read_yearmonth(yearmonth, all_nc_files)
    if (is.null(env_data)) return(NULL)
    # Join environmental data with occurrence points
    env_values <- stars::st_extract(
      x = env_data,
      at = st_coordinates(occ_points))
    occ_data <- bind_cols(occ_points, env_values)
    return(occ_data)
  })
df_env_background <- do.call(rbind, env_background)

# Combine all observations for training
df_enf <- bind_rows(
  mutate(env_background, class = 1),
  mutate(env_background, class = 0))
```

Finally, we use the relatively established `tidysdm` workflow to create a number of ensemble models based on our data and plot a prediction

```r
# Ensemble Modeling and Prediction for the SDM Pipeline

# Variable Selection:
# Select variables with at least 10% contrast between presence and background,
# then remove collinear predictors (cutoff = 0.8) to ensure a robust model.
r_selected <- df_env %>%
  dist_pres_vs_bg(class) %>%
  .[. >= 0.1] %>%
  { r[[names(.)]] } %>%
  filter_collinear(cutoff = 0.8, method = 'cor_caret') %>%
  { r[[.]] }
df_selected <- df_env %>% select(class, names(r_selected))

# Model Fitting:
# Create a recipe and check that "presence" is set as the reference level.
sdm_recipe <- recipe(df_selected, formula = class ~ .)
df_selected %>% check_sdm_presence(class)

# Create a workflow set with multiple SDM algorithms (GLM, RF, GBM, Maxent)
sdm_models <- workflow_set(
  preproc = list(default = sdm_recipe),
  models = list(
    glm = sdm_spec_glm(),
    rf = sdm_spec_rf(),
    gbm = sdm_spec_boost_tree(),
    maxent = sdm_spec_maxent()),
  cross = TRUE) %>%
  option_add(control = control_ensemble_grid())

# Spatial cross-validation using spatial blocks
set.seed(42)
sdm_cv <- spatial_block_cv(data = df_selected, v = 3, n = 5)
autoplot(sdm_cv)

# Tune the models using grid search over the workflows
set.seed(42)
sdm_models <- sdm_models %>%
  workflow_map(
    fn = 'tune_grid',
    resamples = sdm_cv,
    grid = 3,
    metrics = sdm_metric_set(),
    verbose = TRUE)
autoplot(sdm_models)

# Build an ensemble model based on ROC AUC performance
sdm_ensemble <- simple_ensemble() %>%
  add_member(sdm_models, metric = 'roc_auc')
autoplot(sdm_ensemble)

# Final Prediction:
# Use the ensemble model to predict across the final selected environmental raster stack
final_prediction <- sdm_ensemble %>%
  predict_raster(r_selected)

# Plot the final prediction alongside presence and background points
ggplot() +
  geom_spatraster(data = final_prediction, aes(fill = mean)) +
  scale_fill_viridis_c(name = 'Prediction') +
  geom_sf(data = df_selected %>%
            filter(class == 'presence'),
          colour = 'red', alpha = 1) +
  geom_sf(data = df_selected %>%
            filter(class == 'background'),
          colour = 'black', alpha = 0.1) +
  theme_minimal() +
  ggtitle("Final Ensemble SDM Prediction")
```