TFM: Análisis predictivo de incidentes navales en EEUU, 2002 - 2015

Anexo 3.2. Preprocesado Weather Ocean

Oscar Antón

diciembre de 2023

Carga de librerías

```
# Librería                          # Propósito
library(tictoc)                     # Monitorización de tiempo de cómputo
library(progress)                   # Monitorización de tiempo de cómputo

library(leaflet)                    # Visualización geográfica

library(purrr)                      # Programación funcional
library(data.table)                 # Manejo eficiente de conjuntos de datos
library(tidyverse)                  # Sintaxis para el manejo de datos. Incluye dplyr, ggplo
t2, etc.
```

Switches

```
# Guardar datos o no
save_switch <- 0
# Utilizado para no tener que procesar todos los datos en cada renderizado
```

---

1. Adquisición de datos

1.1. Descarga de los datos mensuales de meteorologia maritima de la web de la NOAA

National Oceanic and Atmospheric Administration, de Estados Unidos

```r
if (save_switch  == 1) {
  # URLbase donde están los archivos
  base_url <- "https://www.ncei.noaa.gov/data/global-marine/archive"
  # Carpeta donde quiero guardarlos
  carpeta_local <- "RawDataWeatherOcean"

  # Verifica si la carpeta local existe, y si no, la crea
  if (!dir.exists(carpeta_local)) {
    dir.create(carpeta_local, recursive = TRUE)
  }

  # Crea una secuencia de años y meses desde 2002 hasta 2015
  anos_meses <- expand.grid(ano = 2002:2015, mes = 1:12)

  # Función para descargar estos archivos
  descargar_archivo <- function(ano, mes, base_url, carpeta_local) {
    # Formatea el nombre del archivo
    nombre_archivo <- sprintf("%d%02d.tar.gz", ano, mes)

    # Combina la URL base con el nombre del archivo
    url_descarga <- paste0(base_url, "/", nombre_archivo)

    # Combina la ruta local con el nombre del archivo
    ruta_local <- file.path(carpeta_local, nombre_archivo)

    # Descarga el archivo
    download.file(url_descarga, destfile = ruta_local)

    # Retorna el nombre del archivo descargado
    return(nombre_archivo)
  }

  # Descarga los archivos utilizando lapply
  archivos_descargados <- lapply(1:nrow(anos_meses), function(i) {
    descargar_archivo(anos_meses$ano[i], anos_meses$mes[i], base_url, carpeta_local)
  })
}
```

1.2. Descompresión y selección de los datos mensuales

Al guardar los data.tables:

· Se seleccionan variables

· Se filtra geográficamente: lat[15:70], long[-180:-45]

· Sin información en las variables seleccionadas

```r
# Define carpeta de origen
carpeta_origen <- "RawDataWeatherOcean"
# Define carpeta de destino
carpeta_destino <- "DataWeatherOcean"

# Verifica si la carpeta local existe, y si no, la crea
if (!dir.exists(carpeta_destino)) {
  dir.create(carpeta_destino, recursive = TRUE)
}
# Se establecen las variables seleccionadas
variables_seleccionadas <- c("STATION", "DATE", "LATITUDE", "LONGITUDE", "PAST_WX", "WIND_
SPEED", "VISIBILITY", "AIR_TEMP", "WAVE_HGT")

if (save_switch  == 1) {
  # Define la función para procesar cada archivo .tar.gz
  procesar_archivo <- function(archivo_tar_gz) {
    tic()

    # Extrae el nombre del archivo con solo numeros
    nombre_sin_extension <- gsub("[^0-9]", "", archivo_tar_gz)

    # Crea un directorio temporal para extraer el contenido del archivo
    temp_dir <- tempdir()

    # Descomprime el archivo .tar.gz en el directorio temporal
    untar(archivo_tar_gz, exdir = temp_dir)

    # Lista los archivos CSV extraídos
    archivos_csv <- list.files(temp_dir, pattern = "\\.csv$", full.names = TRUE)

    # Lee y combina los archivos CSV en un solo data.table
    dt <- rbindlist(lapply(archivos_csv, function(file) {
    fread(file, select = variables_seleccionadas, data.table = TRUE, header = TRUE)
    }), fill=TRUE)

    # Filtra las observaciones con 5 o más NAs
    dt <- dt[rowSums(is.na(dt)) < 5]

    # Filtra las observaciones según coordenadas
    dt <- dt[LATITUDE >= 15 & LATITUDE <= 70 & LONGITUDE >= -180 & LONGITUDE <= -45]

    # Exportación y monitorización de datos
    loggedsave(dt, carpeta_destino)

    # Limpia el directorio temporal
    unlink(temp_dir, recursive = TRUE)

  }

  # Lista de archivos .tar.gz en la carpeta de origen
  archivos <- list.files(carpeta_origen, pattern = "\\.tar\\.gz$", full.names = TRUE)

  # Aplica la función a cada archivo
  map(archivos, procesar_archivo)
```

```
}
```

---

2. Agregación de datos en un data.frame conjunto

2.1. Unión en bruto

Se van a combinar verticalmente todos los data.tables mensuales

```
# Define carpeta de origen
carpeta_origen <- "DataWeatherOcean"
# Define carpeta de destino
carpeta_destino <- "DataWeatherOcean"

if (save_switch  == 1) {
  tic()

  # Lista de archivos .rds en la carpeta
  archivos <- list.files(carpeta_origen, pattern = "\\.rds$", full.names = TRUE)

  # Leer y combinar los data.tables verticalmente
  estaciones_maritimas_combinado_1 <- map_df(archivos, readRDS)

  # Exportación y monitorización de datos
  loggedsave(estaciones_maritimas_combinado_1, carpeta_destino)
}
```

2.2. Cálculo de datos diarios

Se van a promediar varias lecturas del mismo día para obtener una única observación por estación y día

```
if (save_switch  == 1) {
  tic()

  # Creamos una función para calcular la moda para aplicar a valores no numéricos
  # Si todos los valores son NA, será NA
  moda <- function(x) {
    if (all(is.na(x))) {
      return(NA)
      }
    frecuencia <- table(x)
    moda <- names(frecuencia)[which.max(frecuencia)]
    return(moda)
  }

  # Agrupamos las observaciones por estación / día
  # Calcular la media diaria para las variables numéricas
  # Aplicar la función de moda al resto de variables
  # Cambiamos de NaN a NA y redondeamos a 3 decimales
  estaciones_maritimas_diario_2 <- estaciones_maritimas_combinado_1 %>%
    mutate(PAST_WX = as.factor(PAST_WX)) %>%
    group_by(STATION, DATE = as.Date(DATE)) %>%
    summarize(across(everything(),
                     ~ ifelse(is.numeric(.), mean(., na.rm = T), moda(.)))) %>%
    ungroup() %>%
    mutate_if(is.numeric, ~ ifelse(is.nan(.), NA, round(., 3))) %>%
    mutate(PAST_WX = as.factor(PAST_WX))

  # Exportación y monitorización de datos
  loggedsave(estaciones_maritimas_diario_2, carpeta_destino)
}
```

---

3. Combinación con Activity_id

```
# Cargar el dataframe de estaciones marítimas
estaciones_maritimas_diario_2 <- as.data.table(readRDS("DataWeatherOcean/estaciones_mariti
mas_diario_2.rds"))

# Cargar el dataframe de EventsOcean
EventsOcean <- as.data.table(readRDS("DataCasualtyAndPollution/Events.rds")) %>%
  filter(watertype == "ocean") %>%
  select(activity_id, date, longitude, latitude) %>%
  unique() %>%
  rename_all(toupper)

# Define carpeta de destino
carpeta_destino <- "DataWeatherOcean"
```

```
if (save_switch  == 1) {
  # Utilizar un enfoque de paralelización multihilo
  library(future)
  plan("multicore")

  tic()
  # Barra de progreso
  pb <- progress_bar$new(total = nrow(EventsOcean ), format = "[:bar] :percent :eta")

  observacion_cercana <- function(incidente) {
    pb$tick()
    # Seleccionar datos correspondientes al Activity_id
    coord_incident <- EventsOcean [ACTIVITY_ID == incidente][1]
    # Seleccionar las observaciones meteorológicas cercanas (+/-2º)
    coord_station <- estaciones_maritimas_diario_2[
      DATE == coord_incident$DATE &
        between(LATITUDE, coord_incident$LATITUDE - 2, coord_incident$LATITUDE + 2) &
        between(LONGITUDE, coord_incident$LONGITUDE - 2, coord_incident$LONGITUDE + 2)
    ]

    # Calcular distancias usando una aproximación
    coord_station[, station_dist := sqrt((LATITUDE - coord_incident$LATITUDE)^2 + (LONGITU
DE - coord_incident$LONGITUDE)^2)]
    # Devolver la observación con la minima distancia
    return(coord_station[order(station_dist)][1][, ACTIVITY_ID := incidente])
  }

  # Aplicar la función a todos los incidentes
  WeatherOcean <- EventsOcean [, purrr::map_df(ACTIVITY_ID, observacion_cercana)]

  # Define carpeta de destino
  carpeta_destino <- "DataWeatherOcean"

  # Exportación y monitorización de datos
  loggedsave(WeatherOcean, carpeta_destino)
}
```

---

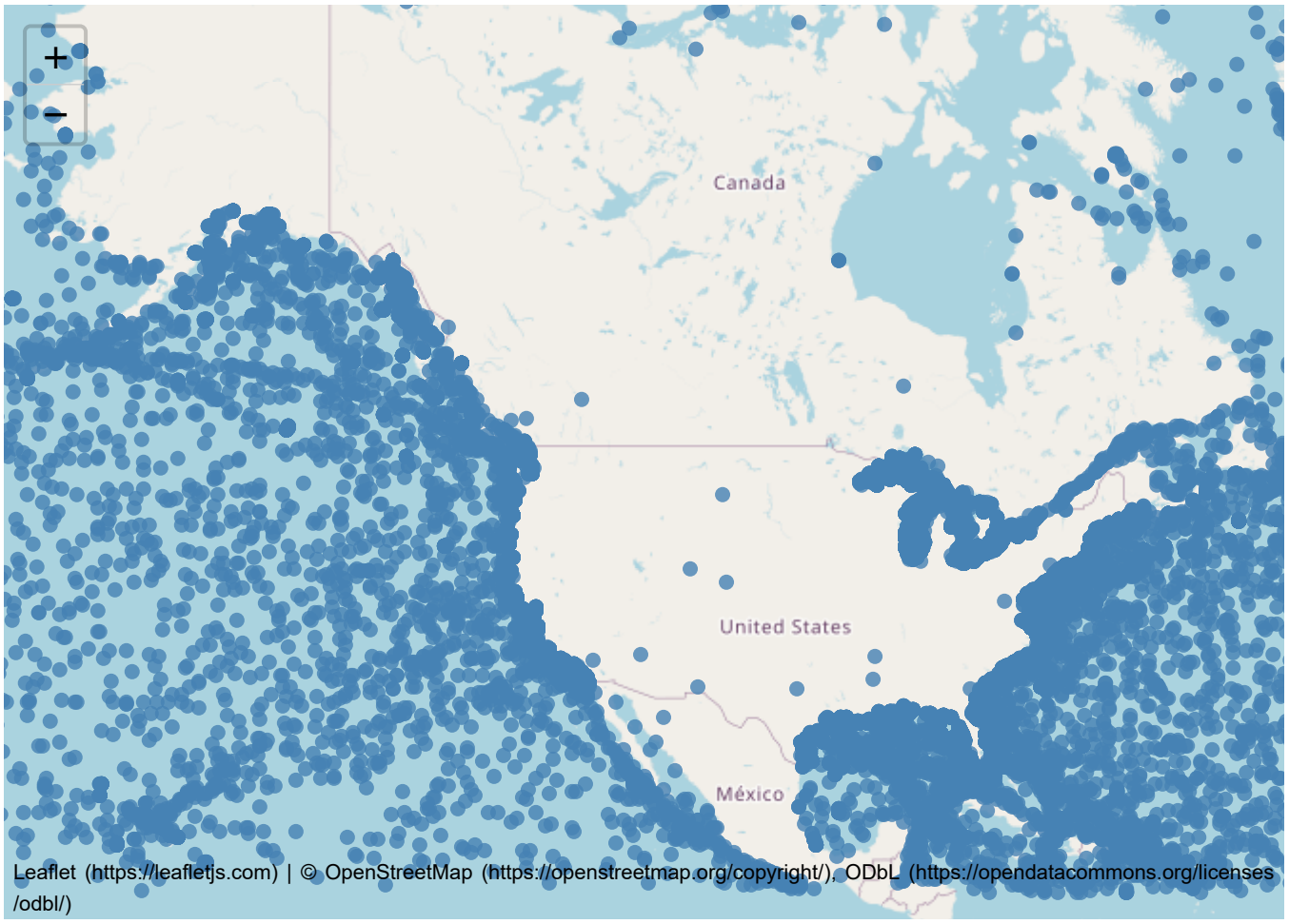
4. Verificación: representación geográfica

4.1. Datos metereológicos

```
# Cargar el dataframe de estaciones marítimas
estaciones_maritimas_diario_2 <- as.data.table(readRDS("DataWeatherOcean/estaciones_mariti
mas_diario_2.rds"))
```

```
# Crear el mapa con una pequeña muestra aleatoria para no saturar la salida
leaflet(estaciones_maritimas_diario_2 %>% sample_frac(0.005)) %>%
  setView(lng = -112, lat = 48, zoom = 3) %>%
  addTiles() %>%
  addCircleMarkers(
    radius = 4,
    popup=~paste("Station:", STATION, "<br>",
                 "Date:", DATE, "<br>",
                 "longitude:", LONGITUDE, "<br>",
                 "latitude:", LATITUDE, "<br>",
                 "Past wx:", PAST_WX, "<br>",
                 "Wind Speed:", WIND_SPEED, "<br>",
                 "Visibility:", VISIBILITY, "<br>",
                 "Air temp:", AIR_TEMP, "<br>",
                 "Wave Hgt:", WAVE_HGT, "<br>"
                 ),
    fillOpacity = 0.8,
    color = "steelblue",
    stroke = FALSE
  )
```

```
## Assuming "LONGITUDE" and "LATITUDE" are longitude and latitude, respectively
```



4.2. Datos meteorológicos con activity_id

```
# Cargar los datos
WeatherOcean <- readRDS("DataWeatherOcean/WeatherOcean.rds")
```

```
# Crear el mapa con una muestra aleatoria para no saturar la salida
leaflet(WeatherOcean %>% sample_frac(0.1)) %>%
  setView(lng = -112, lat = 48, zoom = 3) %>%
  addTiles() %>%
  addCircleMarkers(
    radius = 4,
    popup=~paste("Station:", STATION, "<br>",
                 "Activity id:", ACTIVITY_ID, "<br>",
                 "Date:", DATE, "<br>",
                 "longitude:", LONGITUDE, "<br>",
                 "latitude:", LATITUDE, "<br>",
                 "Past wx:", PAST_WX, "<br>",
                 "Wind Speed:", WIND_SPEED, "<br>",
                 "Visibility:", VISIBILITY, "<br>",
                 "Air temp:", AIR_TEMP, "<br>",
                 "Wave Hgt:", WAVE_HGT, "<br>",
                 "Station dist:", station_dist, "<br>"
                 ),
    fillOpacity = 0.8,
    color = "orange",
    stroke = FALSE
  )
```

```
## Assuming "LONGITUDE" and "LATITUDE" are longitude and latitude, respectively
```

```
## Warning in validateCoords(lng, lat, funcName): Data contains 56 rows with
## either missing or invalid lat/lon values and will be ignored
```