# Configuración del entorno

Sys.setenv("RETICULATE\_PYTHON"="C:/Users/EQUIPO/.ai-navigator/conda/envs/OHW")

# Cargar bibliotecas

library(keras)

library(dplyr)

library(keras3)

# Cargar funciones auxiliares

source("tutorial\_functions.R")

# Leer archivo CSV

data <- readr::read\_csv("data\_FAN.csv")

# Transformación logarítmica de FAN

data <- data %>% mutate(FAN = log1p(FAN))

# Generación de imágenes a partir de datos

image\_list <- make\_image\_list(data,

target\_variable = "FAN",

environmentals = c("TSM", "nivel\_mar", "salinidad\_ssm", "corrientes\_marinas"))

# Separación en conjuntos de entrenamiento y prueba

years <- sapply(image\_list, function(x) { return(x$year) })

image\_list <- split(image\_list, years)

YEARS\_TRAINING <- c("2014", "2016", "2017")

YEARS\_TESTING <- "2015"

train <- pool\_images\_and\_labels(image\_list[YEARS\_TRAINING])

test <- pool\_images\_and\_labels(image\_list[YEARS\_TESTING])

# Construcción del modelo

model <- keras\_model\_sequential() %>%

layer\_dense(units = 64, activation = "relu", input\_shape = dim(train$image)[2]) %>%

layer\_dropout(rate = 0.4) %>%

layer\_dense(units = 32, activation = "relu") %>%

layer\_dropout(rate = 0.3) %>%

layer\_dense(units = 16, activation = "relu") %>%

layer\_dropout(rate = 0.2) %>%

layer\_dense(units = 1, activation = "linear")

summary(model)

# Compilación y entrenamiento

model %>% compile(optimizer = "adam",

loss = "mse",

metrics = c("mae"))

history <- model %>% fit(x = train$image,

y = train$labels,

batch\_size = 128,

epochs = 32,

validation\_split = 0.2,

shuffle = TRUE)

plot(history)

# Evaluación del modelo

metrics <- model %>% evaluate(x = test$image, y = test$labels)

# Predicciones

predictions <- model %>% predict(test$image)

# Matriz de confusión y visualización

results <- tibble(location = test$locations,

date = as.Date(as.numeric(test$dates), origin = "1970-01-01"),

actual\_FAN = test$labels,

predicted\_FAN = predictions)

head(results)

\_\_----\_\_\_\_\_----------\_\_\_\_\_\_\_\_\_--------\_\_\_\_\_\_\_

#' Set classification values

#'

#' @param v vector of numbers (x$FAN)

#' @param lut ordered vector of FAN concentration levels

#' @param na\_value value to replace missing values in v

#' @return ix vector closure codes

#'

#' @export

recode\_classification <- function(v,

lut = c(0,5,15,30),

na\_value = 0){

na <- is.na(v)

v[na] <- na\_value

ix <- findInterval(v, lut) -1

return(ix)

}

#' Normalize FAN and environmental input columns

#'

#' @param x tibble of raw data

#' @param FAN FAN concentration in raw data

#' @param environmentals environmental variables

#' @return tibble of normalized input columns

#'

#' @export

normalize\_input <- function(x, FAN, environmentals) {

image\_cols <- x %>%

dplyr::select(dplyr::all\_of(c(FAN, environmentals)))

other\_cols <- x %>%

dplyr::select(!dplyr::all\_of(c(FAN, environmentals)))

scaling\_factors <- list(min = apply(image\_cols, 2, min, na.rm=TRUE),

max = apply(image\_cols, 2, max, na.rm=TRUE),

mean = apply(image\_cols, 2, mean, na.rm=TRUE),

std = apply(image\_cols, 2, sd, na.rm=TRUE))

scaled\_image\_cols <- sapply(names(image\_cols), function(name) {(image\_cols[[name]] - scaling\_factors$min[name])/(scaling\_factors$max[name] - scaling\_factors$min[name])}, simplify=FALSE) %>%

dplyr::bind\_cols()

scaled\_data <- dplyr::bind\_cols(other\_cols, scaled\_image\_cols)

return(scaled\_data)

}

#' Generate images for FAN concentration forecasting

#'

#' @param raw\_data database with FAN measurements, sampling dates, location, and additional environmental data

#' @param forecast\_steps the number of weeks ahead for forecasting

#' @param n\_steps the number of weeks of samples in an image

#' @param minimum\_gap the smallest gap between samples allowed in an image

#' @param maximum\_gap the largest gap between samples allowed in an image

#' @param FAN variable representing FAN concentration

#' @param environmentals environmental variables

#' @return each list contains an image along with associated metadata

#'

#' @export

make\_image\_list <- function(raw\_data, forecast\_steps, n\_steps, minimum\_gap, maximum\_gap, FAN, environmentals) {

normalized\_data <- raw\_data %>%

dplyr::mutate(classification = recode\_classification(.data[[FAN]]),

meets\_gap = check\_gap(.data$gap\_days, minimum\_gap, maximum\_gap)) %>%

normalize\_input(FAN, environmentals)

find\_images <- function(tbl, key, forecast\_steps, n\_steps, minimum\_gap, maximum\_gap, FAN, environmentals) {

make\_images <- function(batch, tbl, forecast\_steps, n\_steps, minimum\_gap, maximum\_gap, FAN, environmentals) {

image\_batch <- tbl %>% dplyr::slice(batch)

if (any(image\_batch$meets\_gap[2:(n\_steps+forecast\_steps)] == FALSE)) {

z <- list(status=FALSE)

} else {

image <- as.matrix(dplyr::ungroup(image\_batch) %>%

dplyr::select(dplyr::all\_of(c(FAN, environmentals))))

z <- list(status= TRUE,

year = image\_batch$year[1],

location\_id = image\_batch$location\_id[1],

classification = image\_batch$classification[n\_steps+forecast\_steps],

FAN\_concentration = image\_batch[[FAN]][n\_steps+forecast\_steps],

date = image\_batch$date[n\_steps],

image = image[1:n\_steps,])

}

return(z)

}

if (nrow(tbl) < (n\_steps+forecast\_steps)) {

return(NULL)

}

nbatches <- n\_batches(nrow(tbl), (n\_steps+forecast\_steps))

batches <- compute\_batches(nbatches, (n\_steps+forecast\_steps))

xx <- lapply(batches, make\_images, tbl, forecast\_steps, n\_steps, minimum\_gap, maximum\_gap, FAN, environmentals)

gap\_verified <- sapply(xx, function(x){return(x$status)})

xx <- xx[gap\_verified]

return(xx)

}

image\_list <- normalized\_data %>%

dplyr::group\_by(.data$location\_id, .data$year) %>%

dplyr::arrange(date) %>%

dplyr::group\_map(find\_images, forecast\_steps, n\_steps, minimum\_gap, maximum\_gap, FAN, environmentals, .keep=TRUE) %>%

unlist(recursive = FALSE)

return(image\_list)

}

…………………………………………………………..