

Informe Competencia AgTech GitHubJotaPe

Juan Pablo FERRANDEZ

Contents

Resumen	2
Análisis exploratorio	2
Diferencias entre Id y GlobalId	3
Chequeo de GlobalId's repetidos	4
Distribución de clases	4
Anti Join para ver los IDs que no tienen etiquetas	4
Ver si algún punto tiene 2 clasificaciones diferentes	5
Separación de la data en campañas	6
Nueva distribución de las clases	6
Separación por pocas observaciones	7
Mapa para los “pocos datos” de la campaña 18/19	9
Mapa para los “pocos datos” de la campaña 19/20	9
Armando de train y valid	9
Pasaje de R a PYTHON	14
Procesado en PYTHON	14
Idea general de las ventanas de tiempo.	16
LANDSAT (LANDSAT/LC08/C01/T1_SR)	16
SENTINEL (COPERNICUS/S2) Sin Surface Reflectance	19
Cálculos de índices	22
Ejemplo de procesado para un punto.	32
TRAIN campaña 19/20	34
VALID campaña 18/19	35
VALID campaña 19/20	35
TEST campaña 18/19	36
TEST campaña 19/20	37
Generación de los modelos en R	38
Framework H2O	39
Carga de h2o	39
Seteo de variables predictoras y de respuesta	40
Implementación del algoritmo Random Forest	40
Implementación del algoritmo XGBoost (Extreme Gradient Boosting)	42
ENSEMBLE de modelos	44
Guardado y carga de los modelos	45
Dataset de test, y predicciones	45

Armado del submit	45
Guardado del csv para submit	46

Resumen

El presente informe intenta explicar los pasos realizados para generar el mismo archivo `.csv` de submit, que se subió el domingo 13/12/2020, con el que se obtuvo un puntaje público de **0,62262**.

Se comienza con un análisis exploratorio en **R**, con la data original del concurso. La idea general para el entrenamiento de los algoritmos, es separar por campañas, y utilizar un 70% para TRAIN, y el restante 30% para VALID. También en esta etapa, se utiliza la idea de *A. Campos*, de agregar puntos cercanos para aumentar los casos (pero solo en las clases que poseen pocas observaciones). De esos puntos, los *originales*, se usaron en el dataset de validación, y los restantes, se pusieron en el de entrenamiento.

Luego se sigue con un procesamiento de imágenes y features en **Python** con **Google Earth Engine**, basados en las presentaciones de *S. Banchemo*, y creación de índices obtenidos de la pagina recomendada por *Y. Bellini* en su webinar (<https://eos.com/landviewer>). También en esta etapa, se crean **4 ventanas de tiempo**, para tratar de diferenciar las métricas de agrupamiento para 8 colecciones de imágenes diferentes (4 para L8 y 4 para S2).

Ventanas:

- La **ventana 1**, va desde la semana **36**, a la **36** del siguiente año; para tratar de separar las **campanas**.
- La **ventana 2**, va desde la semana **36**, a la **10** del siguiente año; para hacer foco en las **siembras**.
- *Hubo un intento de ventana 3, desde la semana 10, a la 36 del mismo año; para hacer foco en las cosechas, pero no se utilizó porque había un error de procesamiento en test de la campaña 19/20.*
- La **ventana 4**, va desde la semana **52**, a la **12** del siguiente año; para hacer foco en el **forraje a full**.
- La ultima **ventana 5**, es desde la semana **36**, a la **52** del mismo año; para enfocarse en el periodo de **siembra, pero sin el forraje a full**.

Terminada esta etapa de procesamiento de imágenes y features de índices, se pasa nuevamente a **R** para la utilización del framework **h2o** para el entrenamiento de los algoritmos **Random Forest**, y **XGBoost**. Para terminar, se hace un ensemble de estos 2 algoritmos, y se obtienen las predicciones del dataset de test, para luego armar el submit que fue subido a la competencia.

NOTAS: *Los parámetros de los algoritmos se obtuvieron luego de un grid search, pero no se pudo hacer el grid search en este informe, ya que arrojaba un error de h2o. Tampoco se procesan en python todos los registros de los datasets. Solo se procesan 3 registros por cada uno de ellos; pero se deja comentado la forma para poder generar todo el procesamiento. En la etapa de entrenamiento, se cargan los datasets completos, generados con anterioridad.*

Análisis exploratorio

Carga de la data original

```
data_test <- read_csv("../data/data_test.csv")
data_train <- read_csv("../data/data_train.csv")
Etiquetas <- read_csv("../data/Etiquetas.csv")
```

Función para refactorizar tipo de datos

```
cambiar_tipo <- function(dataset){
  retorno <- dataset %>%
    mutate(
      Campania = as.factor(Campania),
      Dataset = as.factor(Dataset),
```

```

        CLASE = as.factor(Cultivo)
      ) %>%
      select(-c(Cultivo))

    return(retorno)
  }

```

TRAIN

```

data_train <- cambiar_tipo(data_train)
summary(data_train)

```

```

##           Id           Longitud           Latitud           Elevacion
## Min.      : 1.0      Min.      :-62.86      Min.      :-34.38      Min.      : -0.00002
## 1st Qu.:119.0      1st Qu.: -62.10      1st Qu.: -33.87      1st Qu.: 99.85998
## Median :235.0      Median : -61.92      Median : -33.80      Median :103.86926
## Mean    :243.9      Mean    : -61.91      Mean    : -33.80      Mean    :102.92332
## 3rd Qu.:366.8      3rd Qu.: -61.71      3rd Qu.: -33.67      3rd Qu.:109.47998
## Max.    :550.0      Max.    : -61.21      Max.    : -33.46      Max.    :126.77998
##
## Dataset      Campania      GlobalId      CLASE
## BC :561      18/19:294      Min.      : 1.0      S      :344
## BCR:289      19/20:556      1st Qu.: 411.5      M      :210
##                      Median : 758.5      s      : 89
##                      Mean    : 750.6      N      : 82
##                      3rd Qu.:1108.5      P      : 55
##                      Max.    :1455.0      X      : 34
##                      (Other): 36

```

TEST

```

data_test <- cambiar_tipo(data_test)
summary(data_test)

```

```

##           Id           Longitud           Latitud           Elevacion
## Min.      : 1.0      Min.      :-62.81      Min.      :-34.37      Min.      : -0.00002
## 1st Qu.:136.5      1st Qu.: -62.12      1st Qu.: -33.88      1st Qu.: 99.23191
## Median :253.0      Median : -61.95      Median : -33.80      Median :104.23316
## Mean    :251.0      Mean    : -61.91      Mean    : -33.80      Mean    :103.46553
## 3rd Qu.:367.5      3rd Qu.: -61.69      3rd Qu.: -33.65      3rd Qu.:109.91555
## Max.    :549.0      Max.    : -61.31      Max.    : -33.46      Max.    :125.09998
##
## Dataset      Campania      GlobalId      CLASE
## BC :371      18/19:228      Min.      : 2.0      NA's:555
## BCR:184      19/20:327      1st Qu.: 333.0
##                      Median : 726.0
##                      Mean    : 718.3
##                      3rd Qu.:1089.0
##                      Max.    :1454.0

```

Diferencias entre Id y GlobalId

```

data_train %>% group_by(Id, GlobalId) %>% count() %>% arrange(Id) %>% head()

```

```

## # A tibble: 6 x 3
## # Groups:   Id, GlobalId [6]
##       Id GlobalId      n

```

```
##      <dbl>      <dbl> <int>
## 1      1          1      1
## 2      1        551      1
## 3      2        984      1
## 4      3        985      1
## 5      4          4      1
## 6      4        554      1
```

De lo anterior, se ve que hay Id repetidos, pero con distinto GlobalId. Se va a tomar GlobalId como unico ID

Chequeo de GlobalId's repetidos

```
data_train %>% group_by(GlobalId) %>% count() %>% arrange(-n) %>% head()
```

```
## # A tibble: 6 x 2
## # Groups:   GlobalId [6]
##   GlobalId     n
##   <dbl> <int>
## 1      1      1
## 2      4      1
## 3      6      1
## 4      7      1
## 5      9      1
## 6     10      1
```

Se ve que no hay GlobalId's repetidos

Distribución de clases

```
data_train %>% group_by(CLASE) %>% count() %>% arrange(-n)
```

```
## # A tibble: 15 x 2
## # Groups:   CLASE [15]
##   CLASE     n
##   <fct> <int>
## 1 S     344
## 2 M     210
## 3 s      89
## 4 N      82
## 5 P      55
## 6 X      34
## 7 U      12
## 8 B       6
## 9 R       6
## 10 m       4
## 11 A       2
## 12 aa      2
## 13 T       2
## 14 G       1
## 15 S/M      1
```

Anti Join para ver los IDs que no tienen etiquetas

Ya que el submit es por ID de etiquetas

```
data_train %>%
  anti_join(Etiquetas, by = c("CLASE" = "Cultivo"))
```

```
## # A tibble: 1 x 8
##       Id Longitud Latitud Elevacion Dataset Campania GlobalId CLASE
##   <dbl>   <dbl>   <dbl>     <dbl> <fct>   <fct>       <dbl> <fct>
## 1   278    -62.0    -33.6      110. BC     18/19      278 S/M
```

El **GlobalId 278** se elimina ya que no está la CLASE S/M taggeada

```
data_train <- data_train %>% filter(GlobalId != 278)
```

Ver si algún punto tiene 2 clasificaciones diferentes

```
data_train %>% group_by(Longitud, Latitud, Campania) %>% count() %>% arrange(-n) %>% head()
```

```
## # A tibble: 6 x 4
## # Groups:   Longitud, Latitud, Campania [6]
##   Longitud Latitud Campania     n
##   <dbl>   <dbl> <fct>   <int>
## 1   -62.0    -33.8 18/19     2
## 2   -62.9    -34.4 19/20     1
## 3   -62.8    -34.4 19/20     1
## 4   -62.8    -34.3 19/20     1
## 5   -62.8    -34.3 19/20     1
## 6   -62.8    -34.3 19/20     1
```

Se busca el punto que tiene $n = 2$ para ver si las las clases son iguales (en cuyo caso no habría mucho problema)

```
data_train %>% filter(round(Longitud,5) == -62.01045,
                      round(Latitud,5) == -33.78365,
                      Campania == '18/19')
```

```
## # A tibble: 2 x 8
##       Id Longitud Latitud Elevacion Dataset Campania GlobalId CLASE
##   <dbl>   <dbl>   <dbl>     <dbl> <fct>   <fct>       <dbl> <fct>
## 1   227    -62.0    -33.8      95.1 BC     18/19      227 X
## 2   228    -62.0    -33.8      95.1 BC     18/19      228 S
```

Se eliminan los **GlobalId 227** y **228**, ya que tiene 2 CLASES diferentes para la misma campaña ('X' y 'S')

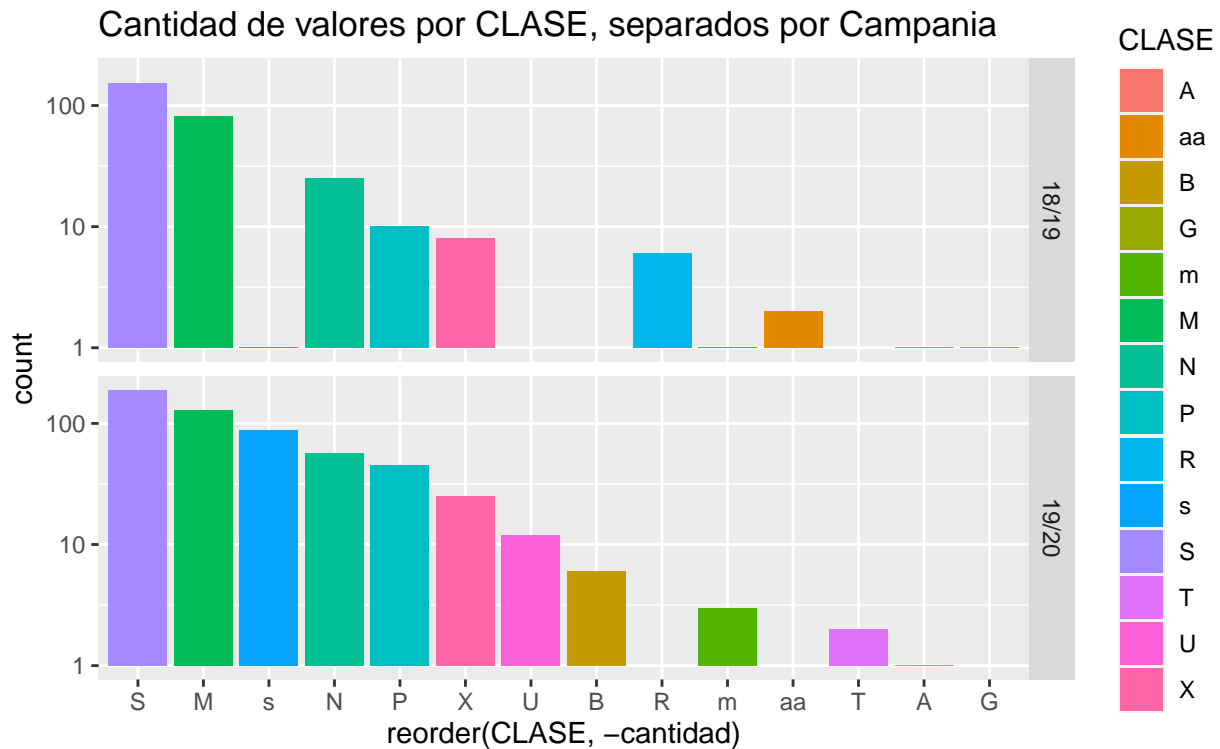
```
data_train <- data_train %>% filter(GlobalId != 227, GlobalId != 228)
```

Gráfico con escala logarítmica de la distribución de clases separados por campaña

```
data_ggplot <- data_train %>%
  select(Dataset, Campania, CLASE, GlobalId) %>%
  group_by(CLASE, Campania) %>%
  mutate(cantidad = n())

data_ggplot %>%
  ggplot(aes(x = reorder(CLASE, -cantidad), fill = CLASE)) +
  geom_bar(stat="count") +
  facet_grid(rows = vars(Campania)) +
  scale_y_log10() +
```

```
labs(title = 'Cantidad de valores por CLASE, separados por Campania')
```



Se observa que en algunas campañas, no hay datos de algunas clases

Separación de la data en campañas

```
separarCampania <- function(paramData, paramCampania){
  retorno <- paramData %>% filter(Campania == paramCampania)
  return(retorno)
}

data_train_18_19 <- separarCampania(data_train, paramCampania = '18/19')
data_train_19_20 <- separarCampania(data_train, paramCampania = '19/20')

data_test_18_19 <- separarCampania(data_test, paramCampania = '18/19')
data_test_19_20 <- separarCampania(data_test, paramCampania = '19/20')
```

Nueva distribución de las clases

```
data_train_18_19 %>% group_by(CLASE) %>% count() %>% arrange(-n)

## # A tibble: 11 x 2
## # Groups:   CLASE [11]
##   CLASE     n
##   <fct> <int>
## 1 S       154
## 2 M        82
## 3 N        25
```

```
## 4 P      10
## 5 X      8
## 6 R      6
## 7 aa     2
## 8 A      1
## 9 G      1
## 10 m     1
## 11 s     1
```

```
data_train_19_20 %>% group_by(CLASE) %>% count() %>% arrange(-n)
```

```
## # A tibble: 11 x 2
## # Groups:   CLASE [11]
##   CLASE     n
##   <fct> <int>
## 1 S     189
## 2 M     128
## 3 s      88
## 4 N      57
## 5 P      45
## 6 X      25
## 7 U      12
## 8 B       6
## 9 m       3
## 10 T       2
## 11 A       1
```

Se ven las clases que tienen 6 o menos observaciones.

18_19

CLASE	CANTIDAD
R	6
aa	2
A	1
G	1
m	1
s	1

19_20

CLASE	CANTIDAD
B	6
m	3
T	2
A	1

Separación por pocas observaciones

```
data_train_pocos_18_19 <- data_train_18_19 %>%
  filter(CLASE %in% c("R", "aa", "A", "G", "m", "s"))
data_train_muchos_18_19 <- data_train_18_19 %>%
  filter(! CLASE %in% c("R", "aa", "A", "G", "m", "s"))
```

```
data_train_pocos_19_20 <- data_train_19_20 %>%
  filter(CLASE %in% c("B", "m", "T", "A"))
data_train_muchos_19_20 <- data_train_19_20 %>%
  filter(! CLASE %in% c("B", "m", "T", "A"))
```

Ver distancia mínima para que no se superpongan los nuevos puntos a crear

```
library(geodist)
library(geosphere)
dMin_18_19 <- georange(cbind(data_train_pocos_18_19$Longitud,
                             data_train_pocos_18_19$Latitud))
cat(c("Distancia minima en campaña 18/19: ",dMin_18_19[1]))
```

```
## Distancia minima en campaña 18/19: 373.749271692646
```

```
dMin_19_20 <- georange(cbind(data_train_pocos_19_20$Longitud,
                             data_train_pocos_19_20$Latitud))
cat(c("Distancia minima en campaña 18/19: ",dMin_19_20[1]))
```

```
## Distancia minima en campaña 18/19: 480.612194781304
```

La distancia mínima es de **370 mts**, así que tomando distancias menores a **150 mts**, no se superpondrían.

Creación de los puntos para Norte, S, E y O, a 90mts para cada uno

#agrego un tag para guardar el punto original, y después pasarlo al set de validación

```
data_train_pocos_18_19$orig <- TRUE
data_train_pocos_19_20$orig <- TRUE
```

```
crearNuevoPuntos <- function(paramData){
  df_nuevos_puntos <- NULL
  distancia = 90

  for (j in 1:nrow(paramData))
  {
    row <- paramData[j,]
    lon <- row$Longitud
    lat <- row$Latitud

    #punto original
    df_nuevos_puntos <- rbind(df_nuevos_puntos, row)

    for(b in seq(0, 359, by = 90)){ #angulos
      nuevo_punto <- geosphere::destPointRhumb(p = c(lon, lat), b = b, d = distancia)
      aux <- row
      aux$Longitud <- nuevo_punto[1]
      aux$Latitud <- nuevo_punto[2]

      aux$orig <- FALSE # se taguea como punto NO original

      #print(row)
      #print(aux)
      df_nuevos_puntos <- rbind(df_nuevos_puntos, aux)
    }
  }
}
```



```

    return(df_nuevos_puntos)
}

#df_nuevos_puntos
data_train_pocos_18_19 <- crearNuevoPuntos(data_train_pocos_18_19)
data_train_pocos_19_20 <- crearNuevoPuntos(data_train_pocos_19_20)

```

Mapa para los “pocos datos” de la campaña 18/19

Para ver si los puntos extras, fueron bien creados.

```

data <- data_train_pocos_18_19
cant_clases <- data %>% group_by(CLASE) %>% select(CLASE) %>% n_distinct

array_colores <- hue_pal()(cant_clases)

data %>%
  leaflet() %>%
  addProviderTiles("CartoDB") %>%
  addCircleMarkers(
    lng = ~Longitud,
    lat = ~Latitud,
    popup = ~GlobalId,
    color = ~array_colores[CLASE],
    radius = 4,
    label = ~CLASE)

```

Imagen de ejemplo del mapa

Mapa para los “pocos datos” de la campaña 19/20

```

data <- data_train_pocos_19_20
cant_clases <- data %>% group_by(CLASE) %>% select(CLASE) %>% n_distinct

array_colores <- hue_pal()(cant_clases)

data %>%
  leaflet() %>%
  addProviderTiles("CartoDB") %>%
  addCircleMarkers(
    lng = ~Longitud,
    lat = ~Latitud,
    popup = ~GlobalId,
    color = ~array_colores[CLASE],
    radius = 4,
    label = ~CLASE)

```

Imagen de ejemplo del mapa

Armando de train y valid

Los puntos extras creados anteriormente, se pasan a TRAIN. Los originales, se pasan a VALID.

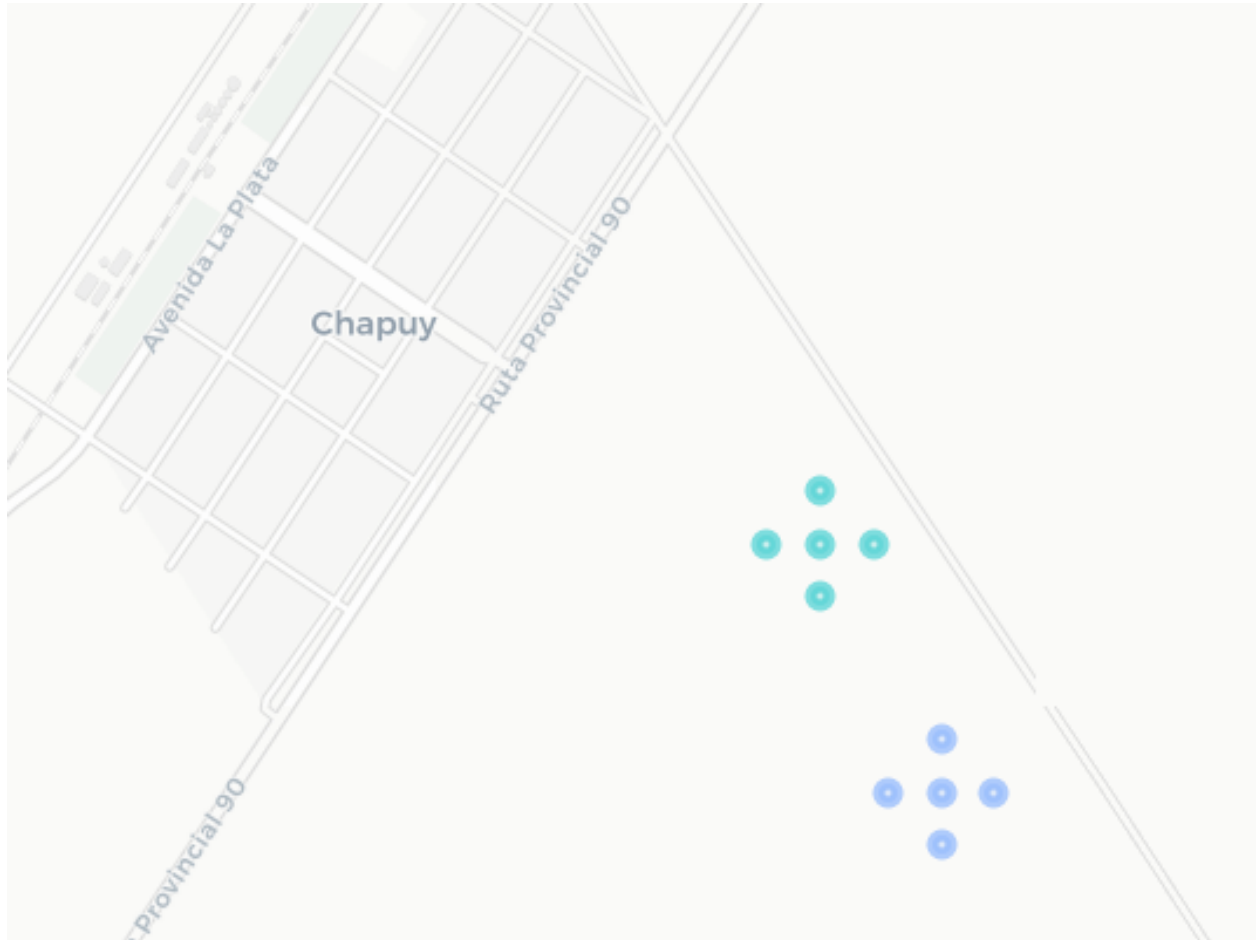


Figure 1: Mapa_18_19



Figure 2: Mapa_19_20

```

data_train_18_19 <- data_train_pocos_18_19 %>% filter(orig == FALSE)
data_valid_18_19 <- data_train_pocos_18_19 %>% filter(! orig == FALSE)

data_train_19_20 <- data_train_pocos_19_20 %>% filter(orig == FALSE)
data_valid_19_20 <- data_train_pocos_19_20 %>% filter(! orig == FALSE)

# eliminacion de la columna para tag
data_train_18_19$orig <- NULL
data_valid_18_19$orig <- NULL

data_train_19_20$orig <- NULL
data_valid_19_20$orig <- NULL

```

También train y valid, pero para las clases con mas casos

```

#18/19
trainIndex <- createDataPartition(data_train_muchos_18_19$CLASE,
                                   p = .7,
                                   list = FALSE,
                                   times = 1)
aux_train <- data_train_muchos_18_19[ trainIndex,]
aux_valid <- data_train_muchos_18_19[-trainIndex,]

#agrego las particiones a los anteriores pocos casos
data_train_18_19 <- rbind(data_train_18_19, aux_train)
data_valid_18_19 <- rbind(data_valid_18_19, aux_valid)

#19/20
trainIndex <- createDataPartition(data_train_muchos_19_20$CLASE,
                                   p = .7,
                                   list = FALSE,
                                   times = 1)
aux_train <- data_train_muchos_19_20[ trainIndex,]
aux_valid <- data_train_muchos_19_20[-trainIndex,]

#agrego las particiones a los anteriores pocos casos
data_train_19_20 <- rbind(data_train_19_20, aux_train)
data_valid_19_20 <- rbind(data_valid_19_20, aux_valid)

```

Distribuciones de train y valid por campañas

```

data_train_18_19 %>% group_by(CLASE) %>% count() %>% arrange(-n)

## # A tibble: 11 x 2
## # Groups:   CLASE [11]
##   CLASE      n
##   <fct> <int>
## 1 S      108
## 2 M       58
## 3 R       24
## 4 N       18
## 5 aa       8
## 6 P        7

```

```
## 7 X      6
## 8 A      4
## 9 G      4
## 10 m     4
## 11 s     4
```

```
data_valid_18_19 %>% group_by(CLASE) %>% count() %>% arrange(-n)
```

```
## # A tibble: 11 x 2
## # Groups:   CLASE [11]
##   CLASE      n
##   <fct> <int>
## 1 S      46
## 2 M      24
## 3 N       7
## 4 R       6
## 5 P       3
## 6 aa      2
## 7 X       2
## 8 A       1
## 9 G       1
## 10 m      1
## 11 s      1
```

```
data_train_19_20 %>% group_by(CLASE) %>% count() %>% arrange(-n)
```

```
## # A tibble: 11 x 2
## # Groups:   CLASE [11]
##   CLASE      n
##   <fct> <int>
## 1 S     133
## 2 M     90
## 3 s     62
## 4 N     40
## 5 P     32
## 6 B     24
## 7 X     18
## 8 m     12
## 9 U      9
## 10 T      8
## 11 A      4
```

```
data_valid_19_20 %>% group_by(CLASE) %>% count() %>% arrange(-n)
```

```
## # A tibble: 11 x 2
## # Groups:   CLASE [11]
##   CLASE      n
##   <fct> <int>
## 1 S     56
## 2 M     38
## 3 s     26
## 4 N     17
## 5 P     13
## 6 X      7
## 7 B      6
## 8 m      3
```

```
## 9 U      3
## 10 T     2
## 11 A     1
```

Pasaje de R a PYTHON

Se utiliza la biblioteca **reticulate** en **R**

```
library(reticulate)
```

```
Sys.setenv(RETICULATE_PYTHON = "/usr/bin/python3.8")
reticulate::py_config()
```

```
## python:      /usr/bin/python3.8
## libpython:   /usr/lib/python3.8/config-3.8-x86_64-linux-gnu/libpython3.8.so
## pythonhome:  //usr://usr
## version:    3.8.5 (default, Jul 28 2020, 12:59:40) [GCC 9.3.0]
## numpy:      /usr/lib/python3/dist-packages/numpy
## numpy_version: 1.17.4
##
## NOTE: Python version was forced by RETICULATE_PYTHON
```

#TRAIN

```
write_csv(x = data_train_18_19, file = paste0('../data/data_train_18_19.csv'))
py$data_train_18_19 <- data_train_18_19
```

```
write_csv(x = data_train_19_20, file = paste0('../data/data_train_19_20.csv'))
py$data_train_19_20 <- data_train_19_20
```

#VALID

```
write_csv(x = data_valid_18_19, file = paste0('../data/data_valid_18_19.csv'))
py$data_valid_18_19 <- data_valid_18_19
```

```
write_csv(x = data_valid_19_20, file = paste0('../data/data_valid_19_20.csv'))
py$data_valid_19_20 <- data_valid_19_20
```

#TEST

```
write_csv(x = data_test_18_19, file = paste0('../data/data_test_18_19.csv'))
py$data_test_18_19 <- data_test_18_19
```

```
write_csv(x = data_test_19_20, file = paste0('../data/data_test_19_20.csv'))
py$data_test_19_20 <- data_test_19_20
```

Procesado en PYTHON

```
import time
import pandas as pd
import numpy as np
import ee
import datetime
```

Inicialización de GEE

```
# La autenticación se hace previamente y se carga Dockerfile
#ee.Authenticate(quiet= True)
```

```
# Inicializar la biblioteca
ee.Initialize()
```

Carga de los datos preprocesados en R

```
# TRAIN
pd_data_train_18_19 = pd.read_csv("/home/rstudio/data/data_train_18_19.csv")
pd_data_train_19_20 = pd.read_csv("/home/rstudio/data/data_train_19_20.csv")

# VALID
pd_data_valid_18_19 = pd.read_csv("/home/rstudio/data/data_valid_18_19.csv")
pd_data_valid_19_20 = pd.read_csv("/home/rstudio/data/data_valid_19_20.csv")

# TEST
pd_data_test_18_19 = pd.read_csv("/home/rstudio/data/data_test_18_19.csv")
pd_data_test_19_20 = pd.read_csv("/home/rstudio/data/data_test_19_20.csv")
```

Función que retorna las coordenadas para armar las **ROI**, y que estas abarquen todos los puntos de train y test

```
def getRectangle(paramTrain, paramValid, paramTest):
```

```
    minLng = min(paramTrain["Longitud"].min(),
                  paramValid["Longitud"].min(),
                  paramTest["Longitud"].min()
                )

    minLat = min(paramTrain["Latitud"].min(),
                  paramValid["Latitud"].min(),
                  paramTest["Latitud"].min()
                )

    maxLng = max(paramTrain["Longitud"].max(),
                  paramValid["Longitud"].max(),
                  paramTest["Longitud"].max()
                )

    maxLat = max(paramTrain["Latitud"].max(),
                  paramValid["Latitud"].max(),
                  paramTest["Latitud"].max()
                )

    return minLng, minLat, maxLng, maxLat
```

Regiones de interés, para ambas campañas

```
ROI_18_19 = ee.Geometry.Rectangle(getRectangle(pd_data_train_18_19,
                                                pd_data_valid_18_19,
                                                pd_data_test_18_19))
ROI_19_20 = ee.Geometry.Rectangle(getRectangle(pd_data_train_19_20,
                                                pd_data_valid_19_20,
                                                pd_data_test_19_20))
```

Función para obtener las fechas de las semanas que se quieran elegir

```
def fechasVenta(semIniCamp1, semFinCamp1, semIniCamp2, semFinCamp2):
    ini_18_19 = datetime.datetime.strptime(semIniCamp1 + '-1', "%Y-W%-W").strftime("%Y-%m-%d")
    fin_18_19 = datetime.datetime.strptime(semFinCamp1 + '-1', "%Y-W%-W").strftime("%Y-%m-%d")
    ini_19_20 = datetime.datetime.strptime(semIniCamp2 + '-1', "%Y-W%-W").strftime("%Y-%m-%d")
    fin_19_20 = datetime.datetime.strptime(semFinCamp2 + '-1', "%Y-W%-W").strftime("%Y-%m-%d")

    return [ini_18_19, fin_18_19, ini_19_20, fin_19_20]

# Ejemplo para las semanas 36 del 2018 / 19 y 20
array_fechas = fechasVenta("2018-W36", "2019-W36", "2019-W36", "2020-W36")
print(array_fechas)
```

```
## ['2018-09-03', '2019-09-09', '2019-09-09', '2020-09-07']
```

Idea general de las ventanas de tiempo.

Elegir las imágenes del satélite, para la ROI elegida, entre las fechas calculadas, para las 2 campañas. Y así, armar las colecciones con las imágenes correspondientes.

Siembra y cosecha de Soja en Gral. Lopez (S.Fe)

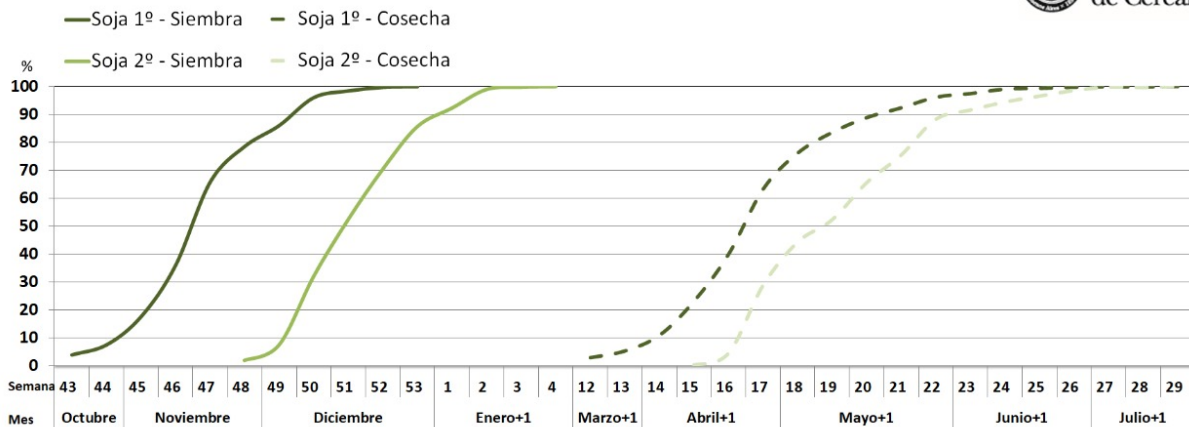


Figure 3: SOJA

LANDSAT (LANDSAT/LC08/C01/T1_SR)

Filtro de nubes

```
# Funcion Banchemo
# https://code.earthengine.google.com/?scriptPath=users%2Fsantiagobanchemo%2Fdesafios-agtech%3AGEE-Ejem

def filtro_nubes(imagen):
    #En esta función utilizamos los bits: 3, 5, 7 y 9
    pixel_qa = imagen.select('pixel_qa')
    bit_mask = 0

    bit_mask += 2 ** 3 # Bit 3: Cloud Shadow
```


Siembra y cosecha de Maíz en Gral. Lopez (S.Fe)

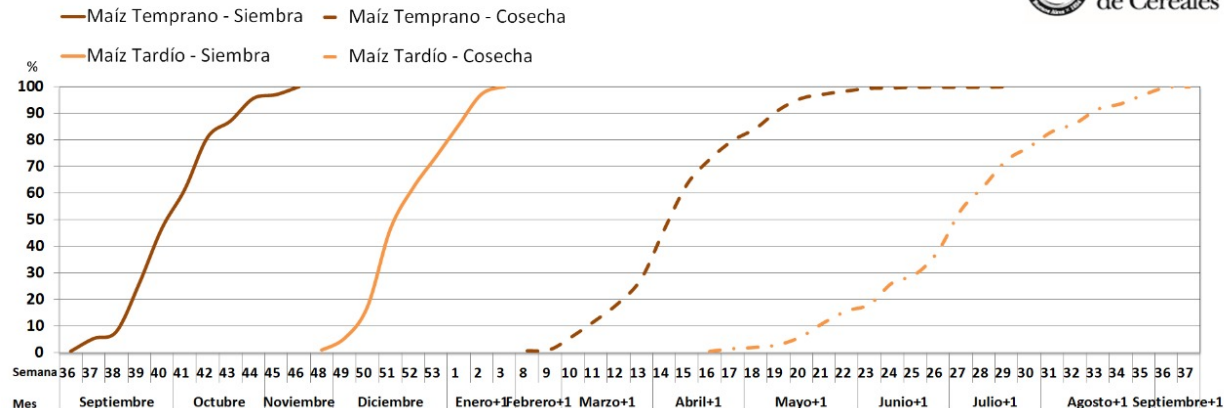


Figure 4: MAIZ

```
bit_mask += 2 ** 5 # Bit 5: Cloud
bit_mask += 2 ** 7 # Bits 6-7: Cloud Confidence
bit_mask += 2 ** 9 # Bits 8-9: Cirrus Confidence

# hacemos un AND entre la banda pixel_qa y nuestra bit_mask
mask = pixel_qa.bitwiseAnd(bit_mask)

retorno = imagen.updateMask(mask.eq(0))

return retorno
```

Ventana de tiempo 1

De la semana 36 a la 36 del siguiente año (como para separar las campañas)

```
#Campaña 18/19
array_fechas = fechasVenta("2018-W36", "2019-W36", "2019-W36", "2020-W36")

v1_coleccion_L8_18_19 = ee.ImageCollection("LANDSAT/LC08/C01/T1_SR") \
    .filterBounds(ROI_18_19) \
    .filterDate(array_fechas[0], array_fechas[1]) \
    .filterMetadata("CLOUD_COVER", "less_than", 30) \
    .map(filtro_nubes)

cantidad_imagenes = v1_coleccion_L8_18_19.size().getInfo()
print("cantidad_imagenes >>> >>> " + str(cantidad_imagenes))

#Campaña 19/20

## cantidad_imagenes >>> >>> 42

v1_coleccion_L8_19_20 = ee.ImageCollection("LANDSAT/LC08/C01/T1_SR") \
    .filterBounds(ROI_19_20) \
    .filterDate(array_fechas[2], array_fechas[3]) \
```

```

        .filterMetadata("CLOUD_COVER", "less_than", 30) \
        .map(filtro_nubes)

cantidad_imagenes = v1_coleccion_L8_19_20.size().getInfo()
print("cantidad_imagenes >>> >>> " + str(cantidad_imagenes))

## cantidad_imagenes >>> >>> 63

```

Ventana de tiempo 2

De la semana **36** a la **10** del siguiente año (como para ver las siembras)

```

array_fechas = fechasVenta("2018-W36", "2019-W10", "2019-W36", "2020-W10")

v2_coleccion_L8_18_19 = ee.ImageCollection("LANDSAT/LC08/C01/T1_SR") \
    .filterBounds(ROI_18_19) \
    .filterDate(array_fechas[0], array_fechas[1]) \
    .filterMetadata("CLOUD_COVER", "less_than", 30) \
    .map(filtro_nubes)

cantidad_imagenes = v2_coleccion_L8_18_19.size().getInfo()
print("cantidad_imagenes >>> >>> " + str(cantidad_imagenes))

## cantidad_imagenes >>> >>> 26

v2_coleccion_L8_19_20 = ee.ImageCollection("LANDSAT/LC08/C01/T1_SR") \
    .filterBounds(ROI_19_20) \
    .filterDate(array_fechas[2], array_fechas[3]) \
    .filterMetadata("CLOUD_COVER", "less_than", 30) \
    .map(filtro_nubes)

cantidad_imagenes = v2_coleccion_L8_19_20.size().getInfo()
print("cantidad_imagenes >>> >>> " + str(cantidad_imagenes))

## cantidad_imagenes >>> >>> 28

```

Ver en el resumen, lo explicado sobre la ventana 3

Ventana de tiempo 4

De la semana **52** a la **12** del siguiente año (para ver el forraje a full). Al ser poco tiempo, es muy posible que en algunos puntos no haya datos de radares.

```

array_fechas = fechasVenta("2018-W52", "2019-W12", "2019-W52", "2020-W12")

v4_coleccion_L8_18_19 = ee.ImageCollection("LANDSAT/LC08/C01/T1_SR") \
    .filterBounds(ROI_18_19) \
    .filterDate(array_fechas[0], array_fechas[1]) \
    .filterMetadata("CLOUD_COVER", "less_than", 40) \
    .map(filtro_nubes)

cantidad_imagenes = v4_coleccion_L8_18_19.size().getInfo()
print("cantidad_imagenes >>> >>> " + str(cantidad_imagenes))

## cantidad_imagenes >>> >>> 14

```

```
v4_coleccion_L8_19_20 = ee.ImageCollection("LANDSAT/LC08/C01/T1_SR") \
    .filterBounds(ROI_19_20) \
    .filterDate(array_fechas[2], array_fechas[3]) \
    .filterMetadata("CLOUD_COVER", "less_than", 30) \
    .map(filtro_nubes)

cantidad_imagenes = v4_coleccion_L8_19_20.size().getInfo()
print("cantidad_imagenes >>> >>> " + str(cantidad_imagenes))

## cantidad_imagenes >>> >>> 18
```

Ventana de tiempo 5

De la semana **36** a la **52** del mismo año (periodo de siembra, pero sin el forraje a full ¿?)

```
array_fechas = fechasVenta("2018-W36", "2018-W52", "2019-W36", "2019-W52")
```

```
v5_coleccion_L8_18_19 = ee.ImageCollection("LANDSAT/LC08/C01/T1_SR") \
    .filterBounds(ROI_18_19) \
    .filterDate(array_fechas[0], array_fechas[1]) \
    .filterMetadata("CLOUD_COVER", "less_than", 40) \
    .map(filtro_nubes)

cantidad_imagenes = v5_coleccion_L8_18_19.size().getInfo()
print("cantidad_imagenes >>> >>> " + str(cantidad_imagenes))

## cantidad_imagenes >>> >>> 18
```

```
v5_coleccion_L8_19_20 = ee.ImageCollection("LANDSAT/LC08/C01/T1_SR") \
    .filterBounds(ROI_19_20) \
    .filterDate(array_fechas[2], array_fechas[3]) \
    .filterMetadata("CLOUD_COVER", "less_than", 30) \
    .map(filtro_nubes)

cantidad_imagenes = v5_coleccion_L8_19_20.size().getInfo()
print("cantidad_imagenes >>> >>> " + str(cantidad_imagenes))

## cantidad_imagenes >>> >>> 13
```

SENTINEL (COPERNICUS/S2) Sin Surface Reflectance

Sin Surface Reflectance, entraban mas imágenes en las colecciones

Filtro de nubes

```
# desde:
# https://developers.google.com/earth-engine/datasets/catalog/COPERNICUS_S2
def maskS2clouds(image):
    qa = image.select('QA60')

    mask = 0
    #Bits 10 and 11 are clouds and cirrus, respectively.
    mask += 2 ** 10
    mask += 2 ** 11
```

```
return image.updateMask(mask)
```

Ventana de tiempo 1

Ídem LANDSAT

```
array_fechas = fechasVenta("2018-W36", "2019-W36", "2019-W36", "2020-W36")

v1_coleccion_S2_18_19 = ee.ImageCollection("COPERNICUS/S2") \
    .filterBounds(ROI_18_19) \
    .filterDate(array_fechas[0], array_fechas[1]) \
    .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 10)) \
    .map(maskS2clouds)

cantidad_imagenes = v1_coleccion_S2_18_19.size().getInfo()
print("cantidad_imagenes >>> >>> " + str(cantidad_imagenes))
```

```
## cantidad_imagenes >>> >>> 75
```

```
v1_coleccion_S2_19_20 = ee.ImageCollection("COPERNICUS/S2") \
    .filterBounds(ROI_19_20) \
    .filterDate(array_fechas[2], array_fechas[3]) \
    .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 10)) \
    .map(maskS2clouds)

cantidad_imagenes = v1_coleccion_S2_19_20.size().getInfo()
print("cantidad_imagenes >>> >>> " + str(cantidad_imagenes))
```

```
## cantidad_imagenes >>> >>> 304
```

Ventana de tiempo 2

Ídem LANDSAT

```
array_fechas = fechasVenta("2018-W36", "2019-W10", "2019-W36", "2020-W10")

v2_coleccion_S2_18_19 = ee.ImageCollection("COPERNICUS/S2") \
    .filterBounds(ROI_18_19) \
    .filterDate(array_fechas[0], array_fechas[1]) \
    .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 10)) \
    .map(maskS2clouds)

cantidad_imagenes = v2_coleccion_S2_18_19.size().getInfo()
print("cantidad_imagenes >>> >>> " + str(cantidad_imagenes))
```

```
## cantidad_imagenes >>> >>> 40
```

```
v2_coleccion_S2_19_20 = ee.ImageCollection("COPERNICUS/S2") \
    .filterBounds(ROI_19_20) \
    .filterDate(array_fechas[2], array_fechas[3]) \
    .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 10)) \
    .map(maskS2clouds)
```

```
cantidad_imagenes = v2_coleccion_S2_19_20.size().getInfo()
print("cantidad_imagenes >>> >>> " + str(cantidad_imagenes))
```

```
## cantidad_imagenes >>> >>> 164
```

Ventana de tiempo 4

Ídem LANDSAT

```
array_fechas = fechasVenta("2018-W52", "2019-W12", "2019-W52", "2020-W12")
```

```
v4_coleccion_S2_18_19 = ee.ImageCollection("COPERNICUS/S2") \
    .filterBounds(ROI_18_19) \
    .filterDate(array_fechas[0], array_fechas[1]) \
    .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 10)) \
    .map(maskS2clouds)
```

```
cantidad_imagenes = v4_coleccion_S2_18_19.size().getInfo()
print("cantidad_imagenes >>> >>> " + str(cantidad_imagenes))
```

```
## cantidad_imagenes >>> >>> 18
```

```
v4_coleccion_S2_19_20 = ee.ImageCollection("COPERNICUS/S2") \
    .filterBounds(ROI_19_20) \
    .filterDate(array_fechas[2], array_fechas[3]) \
    .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 10)) \
    .map(maskS2clouds)
```

```
cantidad_imagenes = v4_coleccion_S2_19_20.size().getInfo()
print("cantidad_imagenes >>> >>> " + str(cantidad_imagenes))
```

```
## cantidad_imagenes >>> >>> 83
```

Ventana de tiempo 5

Ídem LANDSAT

```
array_fechas = fechasVenta("2018-W36", "2018-W52", "2019-W36", "2019-W52")
```

```
v5_coleccion_S2_18_19 = ee.ImageCollection("COPERNICUS/S2") \
    .filterBounds(ROI_18_19) \
    .filterDate(array_fechas[0], array_fechas[1]) \
    .filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 10)) \
    .map(maskS2clouds)
```

```
cantidad_imagenes = v5_coleccion_S2_18_19.size().getInfo()
print("cantidad_imagenes >>> >>> " + str(cantidad_imagenes))
```

```
## cantidad_imagenes >>> >>> 26
```

```
v5_coleccion_S2_19_20 = ee.ImageCollection("COPERNICUS/S2") \
    .filterBounds(ROI_19_20) \
    .filterDate(array_fechas[2], array_fechas[3]) \
```

```
.filter(ee.Filter.lt('CLOUDY_PIXEL_PERCENTAGE', 10)) \
.map(maskS2clouds)

cantidad_imagenes = v5_coleccion_S2_19_20.size().getInfo()
print("cantidad_imagenes >>> >>> " + str(cantidad_imagenes))

## cantidad_imagenes >>> >>> 88
```

























Cálculos de índices

Obtenidos desde <https://eos.com/landviewer> , en la seccion “Combinaciones de bandas”

Índices para L8

























COMBINACIONES DE BANDAS

16 oct. 2020 Landsat 8
8% 53° 225/084

DEFECTO	PERSONALIZADO
 Natural Color B4, B3, B2	
 Color Infrared (Vegetation) B5, B4, B3	
 NDVI $(B5-B4)/(B5+B4)$	
 SAVI $1.5*(B5-B4)/(B5+B4+0.5)$	
 ARVI $(B5-(B4-1*(B2-B4)))/(B5+(B4-1*(B2-B4)))$	
 EVI $2.5*((B5-B4)/((B5+6*B4-7.5*B2)+1))$	
 GCI B5/B3-1	
 SIPI $(B5-B2)/(B5-B4)$	
 NBR $(B5-B7)/(B5+B7)$	
 Agriculture B6, B5, B2	
 False Color (Urban) B7, B6, B4	
 Land/Water B5, B6, B4	

COMBINACIONES DE BANDAS

16 oct. 2020 Landsat 8
8% 53° 225/084

DEFECTO	PERSONALIZADO
 Land/Water B5, B6, B4	
 Healthy Vegetation B5, B6, B2	
 Vegetation Analysis B6, B5, B4	
 NDWI $(B3-B5)/(B3+B5)$	
 Atmospheric Penetration B7, B6, B5	
 Index Stack $(B3-B6)/(B3+B6), (B5-B4)/(B5+B4), (B3-...$	
 Shortwave Infrared B7, B5, B4	
 Atmospheric Removal B7, B5, B3	
 Snow / Clouds B2, B6, B7	
 NDSI $(B3-B6)/(B3+B6)$	
 Panchromatic B8	
 Thermal Infrared B10	

```

def calcular_indice_L8(paramImg):

    # https://developers.google.com/earth-engine/datasets/catalog/LANDSAT\_LC08\_C01\_T1\_SR#bands
    B1_ultraBlue = paramImg.select('B1').rename("B1_ultraBlue")
    B2_blue = paramImg.select('B2').rename("B2_blue")
    B3_green = paramImg.select('B3').rename("B3_green")
    B4_red = paramImg.select('B4').rename("B4_red")
    B5_nir = paramImg.select('B5').rename("B5_nir")
    B6_swir1 = paramImg.select('B6').rename("B6_swir1")
    B7_swir2 = paramImg.select('B7').rename("B7_swir2")

    #paramImg = paramImg.addBands(B4_red)

    ### NDVI
    NDVI = paramImg.expression ('(NIR - RED) / (NIR + RED)',{
        'NIR': B5_nir,
        'RED': B4_red}).rename("NDVI")
    paramImg = paramImg.addBands(NDVI)

    # In Landsat 8, SAVI = ((Band 5 - Band 4) / (Band 5 + Band 4 + 0.5)) * (1.5).
    SAVI = paramImg.expression (
        '(NIR - RED) / (NIR + RED + L)) * (1 + L)',{
        'NIR': B5_nir,
        'RED': B4_red,
        "L" : 0.5
    }).rename("SAVI")
    paramImg = paramImg.addBands(SAVI)

    #ARVI
    ARVI = paramImg.expression (
        '(B5-(B4-1*(B2-B4)))/(B5+(B4-1*(B2-B4)))',{
        'B5': B5_nir,
        'B4': B4_red,
        'B2' : B2_blue
    }).rename("ARVI")
    paramImg = paramImg.addBands(ARVI)

    #EVI
    EVI = paramImg.expression (
        '2.5*((B5-B4)/((B5+6*B4-7.5*B2)+1))',{
        'B5': B5_nir,
        'B4': B4_red,
        'B2' : B2_blue
    }).rename("EVI")
    paramImg = paramImg.addBands(EVI)

    #GCI
    GCI = paramImg.expression (
        'B5/B3-1',{
        'B5': B5_nir,
        'B3' : B3_green
    }).rename("GCI")

```



```

paramImg = paramImg.addBands(GCI)

#SIPI
SIPI = paramImg.expression (
    '(B5-B2)/(B5-B4)',{
        'B5': B5_nir,
        'B4': B4_red,
        'B2' : B2_blue
    }).rename("SIPI")
paramImg = paramImg.addBands(SIPI)

#NBR
NBR = paramImg.expression (
    '(B5-B7)/(B5+B7)',{
        'B5': B5_nir,
        'B7': B7_swir2
    }).rename("NBR")
paramImg = paramImg.addBands(NBR)

#NDWI
NDWI = paramImg.expression (
    '(B3-B5)/(B3+B5)',{
        'B5': B5_nir,
        'B3': B3_green
    }).rename("NDWI")
paramImg = paramImg.addBands(NDWI)

#NDSI
NDSI = paramImg.expression (
    '(B3-B6)/(B3+B6)',{
        'B6': B6_swir1,
        'B3': B3_green
    }).rename("NDSI")
paramImg = paramImg.addBands(NDSI)






bandas = ['B1', 'B2', 'B3', 'B4', 'B5', 'B6', 'B7', 'B10', 'B11',
    'sr_aerosol', 'pixel_qa',
    'NDVI', 'SAVI', 'ARVI', 'EVI', 'GCI', 'SIPI', 'NBR', 'NDWI', 'NDSI']
bandas_r = ["L8_" + sub for sub in bandas]



























return paramImg.select(bandas).rename(bandas_r)

```






Índices para S2



























COMBINACIONES DE BANDAS

 29 nov. 2020  Sentinel-2 L2A
 0%  64°  21HTB

DEFECTO	PERSONALIZADO
 Natural Color B04, B03, B02	
 Forestry Coverage Band B04, B03, B02	
 Fire Detection Band PR_FD	
 Color Infrared (Vegetation) B08, B04, B03	
 NDVI $(B8A-B04)/(B8A+B04)$	
 SAVI $1.5 \cdot (B8A-B04)/(B8A+B04+0.5)$	
 ARVI $(B08-(B04-1 \cdot (B02-B04)))/(B08+(B04-1 \cdot (B02-B04)))$	
 EVI $2.5 \cdot ((B8A-B04)/((B8A+6 \cdot B04-7.5 \cdot B02)+0.5))$	
 GCI B08/B03-1	
 SIPI $(B08-B02)/(B08-B04)$	
 NBR $(B8A-B12)/(B8A+B12)$	
 Agriculture B11, B8A, B02	
 False Color (Urban) B12, B11, B04	

COMBINACIONES DE BANDAS

 29 nov. 2020  Sentinel-2 L2A
 0%  64°  21HTB

DEFECTO	PERSONALIZADO
 Land/Water B8A, B11, B04	
 Healthy Vegetation B8A, B11, B02	
 Vegetation Analysis B11, B8A, B04	
 NDWI $(B03-B08)/(B03+B08)$	
 Atmospheric Penetration B12, B11, B8A	
 Index Stack $(B03-B11)/(B03+B11), (B8A-B04)/(B8A+B04)$	
 Shortwave Infrared B12, B8A, B04	
 Atmospheric Removal B12, B8A, B03	
 Snow / Clouds B02, B11, B12	
 NDSI $(B03-B11)/(B03+B11)$	
 Scene Classification SCL	
 NDVI Classic $(B8A-B04)/(B8A+B04)$	
 Deforestation Index $(B03+B09) \cdot (B11+B12)$	

```

def calcular_indice_S2(paramImg):

    #https://developers.google.com/earth-engine/datasets/catalog/COPERNICUS_S2_SR#bands
    B1_aerosols = paramImg.select('B1').rename("B1_aerosols")
    B2_blue = paramImg.select('B2').rename("B2_blue")
    B3_green = paramImg.select('B3').rename("B3_green")
    B4_red = paramImg.select('B4').rename("B4_red")
    B5_re1 = paramImg.select('B5').rename("B5_re1")
    B6_re2 = paramImg.select('B6').rename("B6_re2")
    B7_re3 = paramImg.select('B7').rename("B7_re3")
    B8_nir = paramImg.select('B8').rename("B8_nir")
    B8A_re4 = paramImg.select('B8A').rename("B8A_re4")
    B9_wv = paramImg.select('B9').rename("B9_wv")
    B11_swir1 = paramImg.select('B11').rename("B11_swir1")
    B12_swir2 = paramImg.select('B12').rename("B12_swir2")

    #NDVI
    NDVI = paramImg.expression ('(NIR - RED) / (NIR + RED)',{
        'NIR': B8A_re4,
        'RED': B4_red}).rename("NDVI")
    paramImg = paramImg.addBands(NDVI)

    ### OTRO NDVI
    NDVI2 = paramImg.expression ('(NIR - RED) / (NIR + RED)',{
        'NIR': B8_nir,
        'RED': B4_red}).rename("NDVI2")
    paramImg = paramImg.addBands(NDVI2)

    #SAVI
    SAVI = paramImg.expression (
        '((NIR - RED) / (NIR + RED + L)) * (1 + L)',{
            'NIR': B8A_re4,
            'RED': B4_red,
            "L" : 0.5
        }).rename("SAVI")
    paramImg = paramImg.addBands(SAVI)

    #SAVI2
    SAVI2 = paramImg.expression (
        '((NIR - RED) / (NIR + RED + L)) * (1 + L)',{
            'NIR': B8_nir,
            'RED': B4_red,
            "L" : 0.5
        }).rename("SAVI2")
    paramImg = paramImg.addBands(SAVI2)

    #ARVI
    ARVI = paramImg.expression (
        '(B8-(B4-1*(B2-B4)))/(B8+(B4-1*(B2-B4)))',{
            'B8': B8_nir,
            'B4': B4_red,
            'B2' : B2_blue

```

```

    }).rename("ARVI")
paramImg = paramImg.addBands(ARVI)

#EVI
EVI = paramImg.expression (
    '2.5*((B8A-B4)/((B8A+6*B4-7.5*B2)+1))',{
        'B8A': B8A_re4,
        'B4': B4_red,
        'B2' : B2_blue
    }).rename("EVI")
paramImg = paramImg.addBands(EVI)

#GCI
GCI = paramImg.expression (
    'B8/B3-1',{
        'B8': B8_nir,
        'B3' : B3_green
    }).rename("GCI")
paramImg = paramImg.addBands(GCI)

#SIPI
SIPI = paramImg.expression (
    '(B8-B2)/(B8-B4)',{
        'B8': B8_nir,
        'B4': B4_red,
        'B2' : B2_blue
    }).rename("SIPI")
paramImg = paramImg.addBands(SIPI)

#NBR
NBR = paramImg.expression (
    '(B8-B12)/(B8+B12)',{
        'B8': B8_nir,
        'B12': B12_swir2
    }).rename("NBR")
paramImg = paramImg.addBands(NBR)

#NDWI
NDWI = paramImg.expression (
    '(B3-B8)/(B3+B8)',{
        'B8': B8_nir,
        'B3': B3_green
    }).rename("NDWI")
paramImg = paramImg.addBands(NDWI)

#NDSI
NDSI = paramImg.expression (
    '(B3-B11)/(B3+B11)',{
        'B11': B11_swir1,
        'B3': B3_green
    }).rename("NDSI")
paramImg = paramImg.addBands(NDSI)

```



```

v2_coleccion_features_S2_19_20 = v2_coleccion_S2_19_20.map(calcular_indice_S2)

# ventana 4 - S2
v4_coleccion_features_S2_18_19 = v4_coleccion_S2_18_19.map(calcular_indice_S2)
v4_coleccion_features_S2_19_20 = v4_coleccion_S2_19_20.map(calcular_indice_S2)

# ventana 5 - S2
v5_coleccion_features_S2_18_19 = v5_coleccion_S2_18_19.map(calcular_indice_S2)
v5_coleccion_features_S2_19_20 = v5_coleccion_S2_19_20.map(calcular_indice_S2)
v5_bandas_imagen_S2_19_20 = v5_coleccion_features_S2_19_20.first().bandNames().getInfo()
print("\nbandas_imagen S2 >>>>> " + str(v5_bandas_imagen_S2_19_20))

##
## bandas_imagen S2 >>>>> ['S2_B1', 'S2_B2', 'S2_B3', 'S2_B4', 'S2_B5', 'S2_B6', 'S2_B7', 'S2_B8', 'S2_B9', 'S2_B10', 'S2_B11', 'S2_B12']

```

Reducción de imágenes

```

# Tambien sacado del ejemplo de Banhero
# https://code.earthengine.google.com/?scriptPath=users%2Fsantiagobanhero%2Fdesafios-agtech%3AGEE-Ejemplos%2F02%20Reduccion%20de%20imagenes

def reduccion(paramColeccion):
    img_mediana = paramColeccion.reduce(ee.Reducer.median())
    img_media = paramColeccion.reduce(ee.Reducer.mean())
    img_min = paramColeccion.reduce(ee.Reducer.min())
    img_max = paramColeccion.reduce(ee.Reducer.max())
    img_std = paramColeccion.reduce(ee.Reducer.stdDev())
    img_amp = img_max.subtract(img_min)

    #se cambia de nombre, sino queda el _max, de la primer imagen
    img_amp = img_amp.regexpRename(regex = '_max', replacement = '_amp', all = True)

    retorno = img_mediana \
        .addBands(img_media) \
        .addBands(img_min) \
        .addBands(img_max) \
        .addBands(img_std) \
        .addBands(img_amp)

    return retorno

```

Aplicación de la reducción para cada ventana

```

# V1 L8
v1_img_features_L8_18_19 = reduccion(v1_coleccion_features_L8_18_19)
v1_img_features_L8_19_20 = reduccion(v1_coleccion_features_L8_19_20)

# V2 L8
v2_img_features_L8_18_19 = reduccion(v2_coleccion_features_L8_18_19)
v2_img_features_L8_19_20 = reduccion(v2_coleccion_features_L8_19_20)

# V4 L8
v4_img_features_L8_18_19 = reduccion(v4_coleccion_features_L8_18_19)
v4_img_features_L8_19_20 = reduccion(v4_coleccion_features_L8_19_20)

# V5 L8
v5_img_features_L8_18_19 = reduccion(v5_coleccion_features_L8_18_19)
v5_img_features_L8_19_20 = reduccion(v5_coleccion_features_L8_19_20)

```

```

# V1 S2
v1_img_features_S2_18_19 = reduccion(v1_coleccion_features_S2_18_19)
v1_img_features_S2_19_20 = reduccion(v1_coleccion_features_S2_19_20)
# V2 S2
v2_img_features_S2_18_19 = reduccion(v2_coleccion_features_S2_18_19)
v2_img_features_S2_19_20 = reduccion(v2_coleccion_features_S2_19_20)
# V4 S2
v4_img_features_S2_18_19 = reduccion(v4_coleccion_features_S2_18_19)
v4_img_features_S2_19_20 = reduccion(v4_coleccion_features_S2_19_20)
# V5 S2
v5_img_features_S2_18_19 = reduccion(v5_coleccion_features_S2_18_19)
v5_img_features_S2_19_20 = reduccion(v5_coleccion_features_S2_19_20)

```

Función getInfoBandas (para obtener la info de un punto específico)

```

def getInfoBandas(paramImagen, paramPunto, paramEscala, paramPrefijo = None):
    # todas las bandas de la imagen
    bandas_imagen = paramImagen.bandNames().getInfo()

    retorno = paramImagen \
        .select(bandas_imagen) \
        .reduceRegion(
            reducer = ee.Reducer.first(),
            geometry = paramPunto,
            scale = paramEscala) \
        .getInfo()

    #renombre de los keys Ej: NDVI -> NDVI_60 (se agrega la escala)
    #y un "prefijo" que se pasa como parámetro
    for k in list(retorno):
        retorno[ paramPrefijo + k + "_" + str(paramEscala) ] = retorno.pop(k)

    return retorno

```

Función get__bandas

Acá se agregan los features a los dataset.

Se pasan como parámetros:

- param_df: Dataset el cual se va a recorrer registro por registro, para obtener la data de esas coordenadas puntuales.
- param_train: Flag para saber si la data es de **TRAIN/VALID** o **TEST**
- param_arrayFeat: Array con las ventanas de tiempo de las cuales se tiene la data de las colecciones. Para cada ventana, obtengo los datos *reducidos* de ese periodo de tiempo.

```

def get_bandas(param_df, param_train, param_arrayFeat):
    cont_int = 0
    pd_aux = pd.DataFrame()

    for index, row in param_df.iterrows():
        try:
            punto = ee.Geometry.Point(float(row['Longitud']),float(row['Latitud']))

            # info del dataset original

```

```

result = pd.Series({'lat': row['Latitud'],
                   'lon': row['Longitud'],
                   'GlobalId': row['GlobalId'], # lo llevo para el submit
                   'CLASE': row['CLASE'] if param_train == True else '',
                   'camp': row['Campania']
                   })

# se recorre cada ventana elegida y se suma al registro original
# Escala = 60, para "romper" un poco la imagen, y obtener datos del punto y alrededores.
# https://developers.google.com/earth-engine/guides/scale
for feat in param_arrayFeat:
    data = getInfoBandas(eval(feat), punto, paramEscala = 60, paramPrefijo = feat[0:3])
    data = {key : round(data[key], 3) for key in data} # redondeo a 3 decimales
    # sumo los features
    result = result.append(pd.Series(data))

pd_aux = pd_aux.append(result, ignore_index = True)

cont_int = cont_int + 1
if (cont_int % 10) == 0 :
    print("cont_int: " + str(cont_int))
except:
    print("Problema con: Lon " + str(row['Longitud']) + " Lat " + str(row['Latitud']) + " y param_tra

return pd_aux

```

Ejemplo de procesamiento para un punto.

Se eligen las ventanas 1, 2, 4 y 5, para L8 y S2, de la campaña 18/19 (ya que se va a usar un punto de la campaña 18/19)

```

array_ventanas = ["v1_img_features_L8_18_19",
                  "v2_img_features_L8_18_19",
                  "v4_img_features_L8_18_19",
                  "v5_img_features_L8_18_19",
                  "v1_img_features_S2_18_19",
                  "v2_img_features_S2_18_19",
                  "v4_img_features_S2_18_19",
                  "v5_img_features_S2_18_19"]

```

Elección de un punto, y llamado a la función.

```

pd_ejemplo = pd_data_train_18_19.sample(n=1, random_state=2021)

start_time = time.time()
ejemplo = get_bandas(param_df = pd_ejemplo,
                    param_train = True,
                    param_arrayFeat = array_ventanas
                    )
print("Tiempo para una fila : " + str(time.time() - start_time) + " segundos")

## Tiempo para una fila :20.53488850593567 segundos
print(ejemplo.transpose())

```

##

0


```
## CLASE                S
## GlobalId             286
## camp                 18/19
## lat                  -33.5838
## lon                  -62.0254
## ...                  ...
## v5_S2_SIPI_max_60    1.732
## v5_S2_SIPI_mean_60   1.161
## v5_S2_SIPI_median_60 1.103
## v5_S2_SIPI_min_60    0.859
## v5_S2_SIPI_stdDev_60 0.302
##
## [1061 rows x 1 columns]
```

Lista con todos los features creados

```
# Primeros 10
```

```
list(ejemplo.keys())[1:11]
```

```
## ['GlobalId', 'camp', 'lat', 'lon', 'v1_L8_ARVI_amp_60', 'v1_L8_ARVI_max_60', 'v1_L8_ARVI_mean_60', 'v1_L8_ARVI_min_60', 'v1_L8_ARVI_stdDev_60', 'v1_L8_ARVI_stdDev_60']
```

```
# Últimos 10
```

```
list(ejemplo.keys())[-10:]
```

```
## ['v5_S2_SAVI_mean_60', 'v5_S2_SAVI_median_60', 'v5_S2_SAVI_min_60', 'v5_S2_SAVI_stdDev_60', 'v5_S2_SAVI_stdDev_60']
```

GENERACIÓN DE LOS DATASETS CON FEATURES

NOTA: Se van a generar solo 3 filas para cada uno de los datasets, por un tema del tiempo de generación

En caso que se quiera generar todo el dataset, comentar las líneas indicadas en el código.

Todos los datasets, son guardados con el postfijo "_3.csv"

```
## TRAIN campaña 18/19
```

```
```python
```

```
Esto es por si se comenta la generación de ejemplos, y se quiere ejecutar el dataset completo; sino,
postfijo = ""
```

```
en caso de querer generar todo el dataset, comentar las 2 siguientes líneas
```

```
postfijo = "_3"
```

```
pd_data_train_18_19 = pd_data_train_18_19.sample(n=3, random_state=2021)
```

```
start_time = time.time()
```

```
pd_train_features_18_19 = get_bandas(param_df = pd_data_train_18_19,
 param_train = True,
 param_arrayFeat = array_ventanas
)
```

```
print("Tiempo de procesamiento :" + str(time.time() - start_time) + " segundos")
```

```
Tiempo de procesamiento :41.12374401092529 segundos
```

```
print(pd_train_features_18_19.transpose())
```

```
0 1 2
CLASE S S M
GlobalId 286 520 281
camp 18/19 18/19 18/19
lat -33.5838 -33.6964 -33.5973
lon -62.0254 -61.5837 -62.0261
...
v5_S2_SIPI_max_60 1.732 1.551 1.326
v5_S2_SIPI_mean_60 1.161 1.255 1.095
v5_S2_SIPI_median_60 1.103 1.229 1.068
v5_S2_SIPI_min_60 0.859 0.964 0.873
v5_S2_SIPI_stdDev_60 0.302 0.152 0.142
##
[1061 rows x 3 columns]
```

Guardado del dataset generado

```
pd_train_features_18_19.to_csv('../data/pd_train_features_18_19'+postfijo+'.csv', index = False, header
```

## TRAIN campaña 19/20

```
postfijo = ""
```

```
en caso de querer generar todo el dataset, comentar las 2 siguientes líneas
```

```
postfijo = "_3"
```

```
pd_data_train_19_20 = pd_data_train_19_20.sample(n=3, random_state=2021)
```

```
start_time = time.time()
```

```
pd_train_features_19_20 = get_bandas(param_df = pd_data_train_19_20,
 param_train = True,
 param_arrayFeat = array_ventanas
)
```

```
print("Tiempo de procesamiento : " + str(time.time() - start_time) + " segundos")
```

```
Tiempo de procesamiento :146.33835172653198 segundos
```

```
print(pd_train_features_19_20.transpose())
```

```
#Guardado del dataset generado
```

```
0 1 2
CLASE X M B
GlobalId 557 967 799
camp 19/20 19/20 19/20
lat -33.63 -33.6253 -33.932
lon -61.8751 -61.3664 -62.0994
...
v5_S2_SIPI_max_60 1.449 1.401 1.023
v5_S2_SIPI_mean_60 1.26 1.267 0.975
v5_S2_SIPI_median_60 1.308 1.301 0.984
v5_S2_SIPI_min_60 0.897 0.894 0.911
v5_S2_SIPI_stdDev_60 0.152 0.133 0.039
##
```

```
[1061 rows x 3 columns]
pd_train_features_19_20.to_csv('../data/pd_train_features_19_20'+postfijo+'.csv',
 index = False,
 header=True)
```

Guardado del dataset generado

```
pd_train_features_19_20.to_csv('../data/pd_train_features_19_20'+postfijo+'.csv',
 index = False,
 header=True)
```

## VALID campaña 18/19

```
postfijo = ""

en caso de querer generar todo el dataset, comentar las 2 siguientes líneas
postfijo = "_3"
pd_data_valid_18_19 = pd_data_valid_18_19.sample(n=3, random_state=2021)

start_time = time.time()
pd_valid_features_18_19 = get_bandas(param_df = pd_data_valid_18_19,
 param_train = True,
 param_arrayFeat = array_ventanas
)
print("Tiempo de procesamiento :" + str(time.time() - start_time) + " segundos")
```

## Tiempo de procesamiento :56.31879281997681 segundos

```
print(pd_valid_features_18_19.transpose())
```

```
0 1 2
CLASE S aa M
GlobalId 106 201 367
camp 18/19 18/19 18/19
lat -33.8679 -33.6302 -33.8097
lon -62.1913 -62.0076 -61.8868
...
v5_S2_SIPi_max_60 1.847 1.042 1.12
v5_S2_SIPi_mean_60 1.173 0.935 0.989
v5_S2_SIPi_median_60 1.048 0.926 0.984
v5_S2_SIPi_min_60 0.818 0.829 0.805
v5_S2_SIPi_stdDev_60 0.347 0.085 0.1
##
[1061 rows x 3 columns]
```

Guardado del dataset generado

```
pd_valid_features_18_19.to_csv('../data/pd_valid_features_18_19'+postfijo+'.csv',
 index = False,
 header=True)
```

## VALID campaña 19/20

```
postfijo = ""
```

```

en caso de querer generar todo el dataset, comentar las 2 siguientes líneas
postfijo = "_3"
pd_data_valid_19_20 = pd_data_valid_19_20.sample(n=3, random_state=2021)

start_time = time.time()
pd_valid_features_19_20 = get_bandas(param_df = pd_data_valid_19_20,
 param_train = True,
 param_arrayFeat = array_ventanas
)
print("Tiempo de procesamiento :" + str(time.time() - start_time) + " segundos")

Tiempo de procesamiento :92.9226541519165 segundos
print(pd_valid_features_19_20.transpose())

0 1 2
CLASE P S m
GlobalId 953 1402 611
camp 19/20 19/20 19/20
lat -33.6654 -33.7553 -33.6695
lon -61.3907 -61.6582 -61.9045
...
v5_S2_SIPI_max_60 1.205 1.306 1.284
v5_S2_SIPI_mean_60 0.968 1.116 1.02
v5_S2_SIPI_median_60 0.925 1.117 0.996
v5_S2_SIPI_min_60 0.877 0.89 0.905
v5_S2_SIPI_stdDev_60 0.099 0.106 0.087
##
[1061 rows x 3 columns]

Guardado del dataset generado

pd_valid_features_19_20.to_csv('../data/pd_valid_features_19_20'+postfijo+'.csv',
 index = False,
 header=True)

```

## TEST campaña 18/19

```

postfijo = ""

en caso de querer generar todo el dataset, comentar las 2 siguientes líneas
postfijo = "_3"
pd_data_test_18_19 = pd_data_test_18_19.sample(n=3, random_state=2021)

start_time = time.time()
pd_test_features_18_19 = get_bandas(param_df = pd_data_test_18_19,
 param_train = False,
 param_arrayFeat = array_ventanas
)
print("Tiempo de procesamiento :" + str(time.time() - start_time) + " segundos")

Tiempo de procesamiento :157.40780639648438 segundos
print(pd_test_features_18_19.transpose())

```

```

0 1 2

```

```
CLASE
GlobalId 147 474 506
camp 18/19 18/19 18/19
lat -33.9352 -33.675 -33.724
lon -62.0976 -61.5177 -61.6242
...
v5_S2_SIPI_max_60 1.514 1.27 1.324
v5_S2_SIPI_mean_60 1.376 1.06 1.153
v5_S2_SIPI_median_60 1.388 1.051 1.175
v5_S2_SIPI_min_60 1.104 0.933 0.914
v5_S2_SIPI_stdDev_60 0.115 0.108 0.105
##
[1061 rows x 3 columns]
```

Guardado del dataset generado

```
pd_test_features_18_19.to_csv('../data/pd_test_features_18_19'+postfijo+'.csv',
 index = False,
 header=True)
```

## TEST campaña 19/20

```
postfijo = ""

en caso de querer generar todo el dataset, comentar las 2 siguientes líneas
postfijo = "_3"
pd_data_test_19_20 = pd_data_test_19_20.sample(n=3, random_state=2021)

start_time = time.time()
pd_test_features_19_20 = get_bandas(param_df = pd_data_test_19_20,
 param_train = False,
 param_arrayFeat = array_ventanas
)
print("Tiempo de procesamiento :" + str(time.time() - start_time) + " segundos")

Tiempo de procesamiento :72.38136029243469 segundos
print(pd_test_features_19_20.transpose())
```

```
0 1 2
CLASE
GlobalId 1049 898 1200
camp 19/20 19/20 19/20
lat -33.5565 -33.6971 -34.0831
lon -61.6883 -61.3345 -62.3579
...
v5_S2_SIPI_max_60 1.525 1.338 0.928
v5_S2_SIPI_mean_60 1.284 1.113 0.909
v5_S2_SIPI_median_60 1.333 1.126 0.91
v5_S2_SIPI_min_60 0.899 0.903 0.889
v5_S2_SIPI_stdDev_60 0.157 0.111 0.011
##
[1061 rows x 3 columns]
```

Guardado del dataset generado

```
pd_test_features_19_20.to_csv('../data/pd_test_features_19_20'+postfijo+'.csv',
 index = False,
 header=True)
```

## Generación de los modelos en R

Levantado de los datasets **completos**

```
pd_train_features_18_19 <- read_csv("../data/pd_train_features_18_19_completo.csv")
pd_train_features_19_20 <- read_csv("../data/pd_train_features_19_20_completo.csv")

pd_valid_features_18_19 <- read_csv("../data/pd_valid_features_18_19_completo.csv")
pd_valid_features_19_20 <- read_csv("../data/pd_valid_features_19_20_completo.csv")

pd_test_features_18_19 <- read_csv("../data/pd_test_features_18_19_completo.csv")
pd_test_features_19_20 <- read_csv("../data/pd_test_features_19_20_completo.csv")
```

Uniones de las campañas

```
data_train_f <- rbind(pd_train_features_18_19, pd_train_features_19_20)
data_valid_f <- rbind(pd_valid_features_18_19, pd_valid_features_19_20)
data_test_f <- rbind(pd_test_features_18_19, pd_test_features_19_20)
```

```
Borrado de datasets sin usar
pd_train_features_18_19 <- NULL
pd_train_features_19_20 <- NULL

pd_valid_features_18_19 <- NULL
pd_valid_features_19_20 <- NULL

pd_test_features_18_19 <- NULL
pd_test_features_19_20 <- NULL
```

Función **crear\_features**: para convertir algunas columnas a tipo categóricas

```
crear_features <- function(dataset){
 retorno <- dataset %>%
 mutate(
 camp = as.factor(camp),
 CLASE = as.factor(CLASE)
)

 return(retorno)
}

aplicacion de la función a los datasets
data_train_f <- crear_features(data_train_f)
data_valid_f <- crear_features(data_valid_f)
data_test_f <- crear_features(data_test_f)
```

Estandarización de features, ya que se comporta mejor en algunos tipos de algoritmos

No se estandarizan: “CLASE”, “GlobalId”, “camp”, “lat” y “lon”

Se muestra un feature para ver si está bien estandarizado. La media en train, debería ser 0, y en valid, debería ser algo “cercano” a 0

## Pre escalado

```
summary(data_train_f$v1_L8_B10_min_60)
```

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
2798 2833 2842 2846 2858 2987
```

## Post escalado

```
library(dataPreparation)
#Columnas para standarizar. Todas menos....
colsAStand <- setdiff(colnames(data_train_f), c("CLASE", "GlobalId", "camp", "lat", "lon"))
scales <- build_scales(data_set = data_train_f , verbose = FALSE, cols = colsAStand)

data_train_f <- fast_scale(data_set = data_train_f, scales = scales, verbose = FALSE)
data_valid_f <- fast_scale(data_set = data_valid_f, scales = scales, verbose = FALSE)
data_test_f <- fast_scale(data_set = data_test_f, scales = scales, verbose = FALSE)
```

train (debería tener media = 0)

```
summary(data_train_f$v1_L8_B10_min_60)
```

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
-2.6183 -0.6916 -0.1962 0.0000 0.6846 7.7858
```

valid (debería tener media aproximadamente = 0)

```
summary(data_valid_f$v1_L8_B10_min_60)
```

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
-1.737523 -0.636564 -0.196180 -0.006316 0.739636 2.005739
```

Se copian los niveles de train a test, ya que TEST se armó con CLASE vacía

```
levels(data_test_f$CLASE) <- levels(data_train_f$CLASE)
```

## Framework H2O

Framework con el que se va a trabajar en la etapa de generación de modelos

## Carga de h2o

Se configura con 4GB de uso de memoria máxima

```
library(h2o)
```

```
h2o.init(max_mem_size="4G")
```

```
##
H2O is not running yet, starting it now...
##
Note: In case of errors look at the following log files:
/tmp/Rtmp1z6JnF/file12975046b3f/h2o_rstudio_started_from_r.out
/tmp/Rtmp1z6JnF/file12922e080b5/h2o_rstudio_started_from_r.err
##
##
Starting H2O JVM and connecting: ... Connection successful!
##
R is connected to the H2O cluster:
```

```
H2O cluster uptime: 2 seconds 454 milliseconds
H2O cluster timezone: Etc/UTC
H2O data parsing timezone: UTC
H2O cluster version: 3.32.0.1
H2O cluster version age: 2 months and 8 days
H2O cluster name: H2O_started_from_R_rstudio_ofp829
H2O cluster total nodes: 1
H2O cluster total memory: 3.56 GB
H2O cluster total cores: 4
H2O cluster allowed cores: 4
H2O cluster healthy: TRUE
H2O Connection ip: localhost
H2O Connection port: 54321
H2O Connection proxy: NA
H2O Internal Security: FALSE
H2O API Extensions: Amazon S3, XGBoost, Algos, AutoML, Core V3, TargetEncoder, Core V4
R Version: R version 4.0.3 (2020-10-10)
```

```
para ocultar la barra de progreso, sino, "ensucia" mucho el reporte
h2o.no_progress()
```

Carga de los datasets como tipo de dato para h2o

```
h2o_train <- as.h2o(data_train_f)
h2o_valid <- as.h2o(data_valid_f)
h2o_test <- as.h2o(data_test_f)
```

## Seteo de variables predictoras y de respuesta

La de respuesta es 'CLASE'; y las predictoras son todas las features creadas, menos: 'CLASE', 'GlobalId', 'lat' y 'lon' (para que no estén incluidas en el entrenamiento de los modelos)

```
y <- "CLASE"
x <- setdiff(names(h2o_train), c(y, 'GlobalId', 'lat', 'lon'))
```

Función **getBalAcc**: para calcular el **balance accuracy** (métrica de evaluación del concurso)

Se predice la clase en el dataset de validación, y luego se calcula el **bacc**

```
library("mlr3measures")

getBalAcc <- function(paramModel, paramData){
 truth <- as.data.frame(paramData$CLASE) # obtención de la clase verdadera
 aux_pred <- h2o.predict(paramModel, paramData) # predicción del dataset
 response <- as.data.frame(aux_pred$predict) # extracción de la columna de predicción
 levels(response$predict) <- levels(truth$CLASE) # copiado de las categorías de la clase

 return(bacc(truth = truth$CLASE, response = response$predict))
}
```

Seteo de variables comunes para los algoritmos

```
nfolds <- 4
seed <- 2021
```

## Implementación del algoritmo Random Forest

h2o.randomForest



<http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/drf.html>

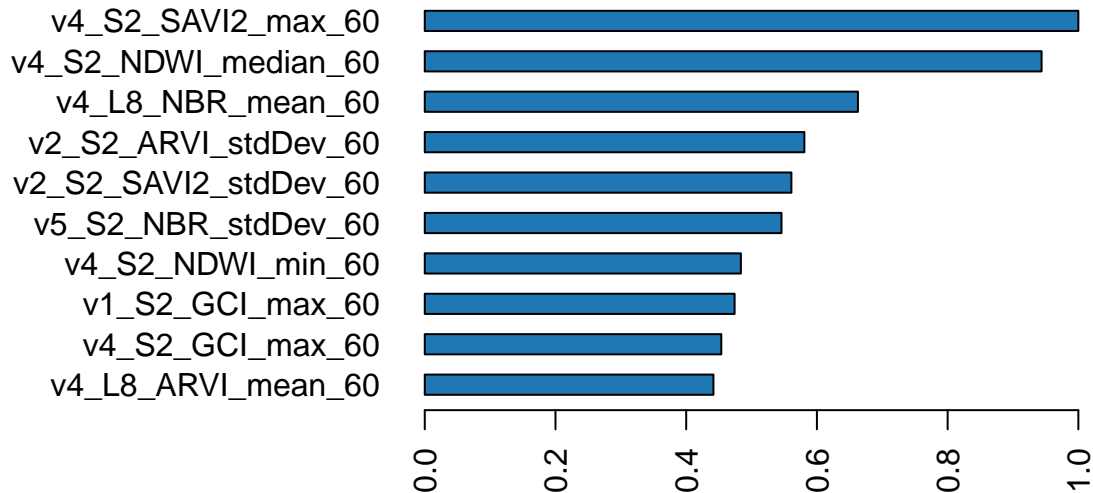
```
model_rf <- h2o.randomForest(
 x = x,
 y = y,
 training_frame = h2o_train,
 nfolds = nfolds,
 keep_cross_validation_models = TRUE,
 keep_cross_validation_predictions = TRUE,
 keep_cross_validation_fold_assignment = FALSE,
 score_each_iteration = FALSE,
 score_tree_interval = 0,
 fold_assignment = "Modulo",
 ignore_const_cols = TRUE,
 balance_classes = FALSE,
 max_after_balance_size = 5,
 #max_confusion_matrix_size = 20,
 ntrees = 50,
 max_depth = 4,
 min_rows = 1,
 nbins = 20,
 nbins_top_level = 1024,
 nbins_cats = 1024,
 r2_stopping = 1.797693e+308,
 stopping_rounds = 0,
 stopping_tolerance = 0.001,
 max_runtime_secs = 0,
 build_tree_one_node = FALSE,
 mtries = -1,
 sample_rate = 0.9,
 binomial_double_trees = FALSE,
 col_sample_rate_change_per_level = 1,
 col_sample_rate_per_tree = 0.3,
 min_split_improvement = 1e-05,
 histogram_type = "UniformAdaptive",
 categorical_encoding = "Enum",
 calibrate_model = FALSE,
 distribution = "multinomial",
 check_constant_response = TRUE,
 gainslift_bins = -1,
 seed = seed + 1
)
```

Plot de la influencia relativa de las variables incluidas en el modelo.

<http://docs.h2o.ai/h2o/latest-stable/h2o-docs/variable-importance.html>

```
h2o.varimp_plot(model_rf)
```

## Variable Importance: DRF



Cálculo del bacc para el dataset de validación

```
getBalAcc(paramModel = model_rf, paramData = h2o_valid)
```

```
[1] 0.8017643
```

## Implementación del algoritmo XGBoost (Extreme Gradient Boosting)

h2o.xgboost

<http://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/xgboost.html>

```
model_xgb <- h2o.xgboost(x = x,
 y = y,
 training_frame = h2o_train,
 nfolds = nfolds,
 keep_cross_validation_models = TRUE,
 keep_cross_validation_predictions = TRUE,
 keep_cross_validation_fold_assignment = FALSE,
 score_each_iteration = FALSE,
 fold_assignment = "Modulo",
 ignore_const_cols = TRUE,
 stopping_rounds = 0,
 stopping_tolerance = 0.001,
 max_runtime_secs = 0,
 distribution = "multinomial",
 tweedie_power = 1.5,
 categorical_encoding = "OneHotInternal",
 quiet_mode = TRUE,
 ntrees = 50,
 max_depth = 50,
```

```

min_rows = 1,
min_child_weight = 1,
learn_rate = 0.05,
eta = 0.05,
sample_rate = 0.7,
subsample = 0.7,
col_sample_rate = 0.3,
colsample_bylevel = 0.3,
col_sample_rate_per_tree = 1,
colsample_bytree = 1,
colsample_bynode = 1,
max_abs_leafnode_pred = 0,
max_delta_step = 0,
score_tree_interval = 0,
min_split_improvement = 0,
gamma = 0,
nthread = -1,
build_tree_one_node = FALSE,
calibrate_model = FALSE,
max_bins = 256,
max_leaves = 0,
sample_type = "uniform",
normalize_type = "tree",
rate_drop = 0,
one_drop = FALSE,
skip_drop = 0,
tree_method = "exact",
grow_policy = "depthwise",
booster = "gbtree",
reg_lambda = 1,
reg_alpha = 0,
dmatrix_type = "dense",
backend = "cpu",
gpu_id = 0,
gainlift_bins = -1,
seed = seed + 3

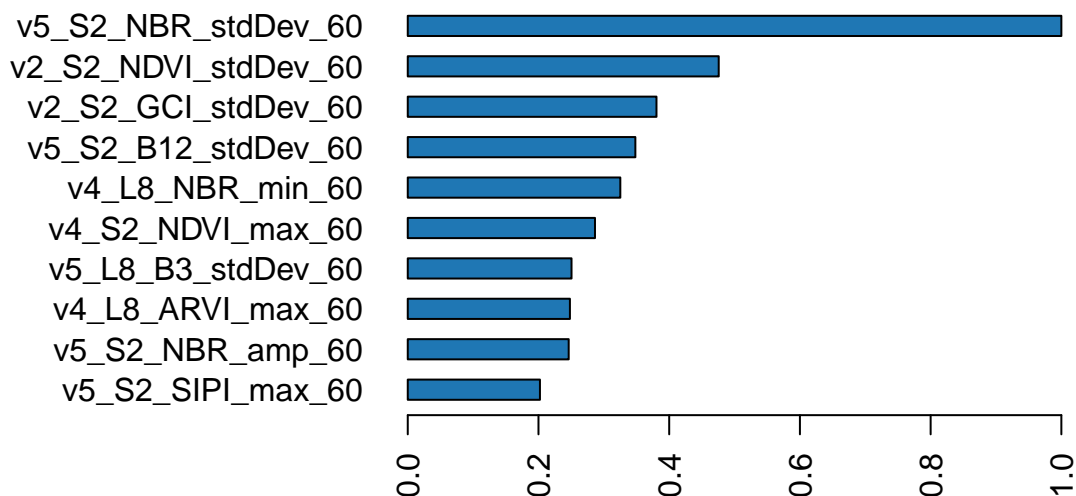
```

)

Plot de la influencia relativa de las variables incluidas en el modelo

```
h2o.varimp_plot(model_xgb)
```

## Variable Importance: XGBOOST



Cálculo del bacc para el dataset de validación

```
getBalAcc(paramModel = model_xgb, paramData = h2o_valid)
```

```
[1] 0.8099828
```

## ENSEMBLE de modelos

Se obtiene el promedio de ambos modelos para ver como funcionan en conjunto, y que no tire el bacc muy para abajo.

También es para generalizar aún mas las predicciones.

```
valid_rf <- h2o.predict(model_rf, h2o_valid)
valid_xgb <- h2o.predict(model_xgb, h2o_valid)

valid_rf$predict <- NULL
valid_xgb$predict <- NULL

valid_ensemble <- (valid_rf + valid_xgb) / 2

claseEnsemble <- colnames(valid_ensemble)[max.col(valid_ensemble, ties.method = "first")]

valid_ensemble$predict <- as.h2o(claseEnsemble)
```

Cálculo del bacc en el ensemble

```
truth <- as.data.frame(h2o_valid$CLASE)
response <- as.data.frame(as.factor(valid_ensemble$predict))
levels(response$predict) <- levels(truth$CLASE)
bacc(truth = truth$CLASE, response = response$predict)
```

```
[1] 0.8058103
```

## Guardado y carga de los modelos

Esto es para saber que efectivamente se están usando los modelos guardados.

Guardado y carga de los modelos, con otro nombre para diferenciarlos

```
path_rf <- h2o.saveModel(object = model_rf, path = "/home/rstudio/modelos/", force = TRUE)
path_xgb <- h2o.saveModel(object = model_xgb, path = "/home/rstudio/modelos/", force = TRUE)

modelo_rf_1 <- h2o.loadModel(path = path_rf)
modelo_xgb_1 <- h2o.loadModel(path = path_xgb)
```

## Dataset de test, y predicciones

```
Random Forest
test_rf_1 <- h2o.predict(modelo_rf_1, h2o_test)

XGBoost
test_xgb_1 <- h2o.predict(modelo_xgb_1, h2o_test)

test_rf_1$predict <- NULL
test_xgb_1$predict <- NULL

ensemble de las predicciones de test
test_ensemble <- (test_rf_1 + test_xgb_1) / 2

claseEnsemble <- colnames(test_ensemble)[max.col(test_ensemble, ties.method = "first")]

test_ensemble$predict <- as.h2o(claseEnsemble)

test_ensemble$GlobalId <- h2o_test$GlobalId
test_ensemble <- test_ensemble[, c("GlobalId", "predict")]

head(test_ensemble)
```

```
GlobalId predict
1 2 M
2 3 M
3 5 M
4 8 N
5 11 N
6 12 P
```

## Armado del submit

Lectura de las etiquetas

```
Etiquetas <- read_csv("../data/Etiquetas.csv")
h2o_Etiquetas <- as.h2o(Etiquetas)
```

Join entre las etiquetas y el ensemble, para obtener los IDs de las clases predichas

```

h2o_Etiquetas$Cultivo <- h2o.asfactor(h2o_Etiquetas$Cultivo)
test_ensemble$predict <- h2o.asfactor(test_ensemble$predict)

submit <- h2o.merge(x = test_ensemble,
 y = h2o_Etiquetas,
 by.x = 'predict',
 by.y = 'Cultivo')

submit$predict <- NULL
submit$Tipo <- NULL
submit <- as.data.frame(submit) %>% arrange(GlobalId)
head(submit)

```

```

GlobalId CultivoId
1 2 3
2 3 3
3 5 3
4 8 10
5 11 10
6 12 9

```

## Guardado del csv para submit

```

write_csv(submit, file = "/home/rstudio/resultado/submit.csv",
 col_names = FALSE,
 append = FALSE)

```

Se comparan los hash del submit que se hizo para el puntaje público, y el generado en este notebook; para ver que los contenidos sean idénticos

```

Archivo generado en la celda anterior
md5sum /home/rstudio/resultado/submit.csv
Archivo subido para el submit
md5sum /home/rstudio/resultado/submit_20201213_62262.csv

```

```

d3a5d302411a237d32f3d813e7390b85 /home/rstudio/resultado/submit.csv
d3a5d302411a237d32f3d813e7390b85 /home/rstudio/resultado/submit_20201213_62262.csv

```

Apagado del framework de H2O

```

h2o.shutdown(prompt = FALSE)

```