

Manual en Español de la librería en R Sparse Kernel methods (SKM)

Authors:

Oswal Antonio Montesinos López^{1*},
Brandon Alejandro Mosqueda González²,
Abel Palafox González³,
Abelardo Montesinos López³,
José Crossa ^{4,5},

¹Facultad de Telemática, Universidad de Colima, Colima, Mexico,

²Centro de Investigación en Computación (CIC), Instituto Politécnico Nacional (IPN), México City, México,

³Centro Universitario de Ciencias Exactas e Ingenierías (CUCEI), Universidad de Guadalajara, Guadalajara, México,

⁴International Maize and Wheat Improvement Center (CIMMYT), Texcoco, México,

⁵Colegio de Postgraduados, Montecillo, México.

Corresponding author: oamontes2@hotmail.com and oamontes1@ucol.mx

Como citar:

Montesinos López OA, Mosqueda González BA, Palafox González A, Montesinos López A and Crossa J (2022) A General-Purpose Machine Learning R Library for Sparse Kernels Methods With an Application for Genome-Based Prediction. Front. Genet. 13:887643. doi: 10.3389/fgene.2022.887643

Table of Contents

1	Conjuntos de datos	4
1.1	ChickpeaToy	4
1.2	EYTTToy	6
1.3	GroundnutToy	8
1.4	MaizeToy.....	10
2	Antes de empezar	12
2.1	Instalación SKM.....	12

2.2	Otras dependencias para los ejemplos.....	12
3	Regresión generalizada.....	12
3.1	Ejemplo de resultados continuos solo G en el predictor con Grid search y particiones aleatorias	12
3.2	Ejemplo con respuesta binaria con Grid search y validación cruzada 7-fold con $env + g$ en el predictor.....	19
3.3	Ejemplo con variable respuesta categórica con optimización bayesiana con partición aleatoria $Env + G + GE$ dentro del predictor	27
3.4	Ejemplo con datos de conteo con optimización bayesiana con línea de partición aleatoria $Env + G + GE$ en el predictor	35
3.5	Ejemplo multivariados con respuestas continuas con optimización bayesiana con validación cruzada 7-fold con $Env + G$ el predictor	42
3.6	Ejemplo con kernels con Grid search y particiones aleatorias	51
3.7	Ejemplo de con Sparse Kernels con Grid search y particiones aleatorias.....	60
4	Métodos de regresión bayesiana.....	69
4.1	Ejemplo de resultados continuos con Bayesian Lasso con solo G en el predictor con Grid search y particiones aleatorias	69
4.2	Ejemplo de con una respuesta binaria con Bayes A $Env + G$ en el predictor con Grid search y validación cruzada 7-fold	73
4.3	Ejemplo con variable respuesta categórica con Bayes C con optimización bayesiana con particiones aleatorias $Env + G + GE$ dentro del predictor	77
4.4	Ejemplo de resultado continuo con GBLUP con optimización bayesiana con línea de partición aleatoria $Env + G + GE$ dentro del predictor	81
4.5	Ejemplo de multivariado con respuestas continuas con regresión Bayesian Ridge Con optimización bayesiana con validación cruzada 7-fold con $Env + G$ el predictor	86
4.6	Ejemplo con kernels con Grid search y particiones aleatorias.....	90
4.7	Ejemplo de con kernels con Grid search y particiones aleatorias	99
5	Random forest	109
5.1	Ejemplo con respuesta continua con Grid search y particiones aleatorias solo G en el predictor	109
5.2	Ejemplo con respuesta binaria con Grid search y validación cruzada 7-fold con $Env + G$ el predictor	121
5.3	Ejemplo con variable respuesta categórica con optimización bayesiana con particiones aleatorias $Env + G + GE$ dentro del predictor	127
5.4	Ejemplo multivariado con respuestas continuas con optimización bayesiana con línea de partición aleatoria $Env + G + GE$ dentro del predictor	134
5.5	Ejemplo multivariado mixto (con respuestas binaria, categórica y continua) Con optimización bayesiana con validación cruzada 7-fold con $*Env+G\$$ en el predictor.....	140

5.6	Ejemplo con kernels con Grid search y particiones aleatorias	157
5.7	Ejemplo con sparse Kernels con Grid search y particiones aleatorias	165
6	Generalized boosted machines methods	174
6.1	Ejemplo con respuesta continua con Grid search y partciones aleatorias con solo G en el 174	
6.2	Ejemplo con respuesta binaria con Grid search y validación cruzada 7-fold con $Env+G$ en el predictor	180
6.3	Ejemplo con variable respuesta categórica con optimización bayesiana con línea de partición aleatoria $Env + G + GE$ dentro del predictor	187
6.4	Ejemplo con kernels con Grid search y particiones aleatorias	193
6.5	Ejemplo de con Sparse Kernels con Grid earch y particiones aleatorias	202
7	Support vector machine methods.....	211
7.1	Ejemplo con respuesta continua con Grid search y particiones aleatorias solo G en el predictor	211
7.2	Ejemplo con respuesta binaria con Grid search y validación cruzada 7-fold con $Env + G$ el predictor	218
7.3	Ejemplo de con respuesta categórica con optimización bayesiana con línea de partición aleatoria $Env + G + GE$ dentro del predictor	224
8	Deep learning.....	230
8.1	Ejemplo con respuesta continua con Grid search y particiones aleatorias solo G en el predictor	230
8.2	Ejemplo con respuesta binaria con Grid search y validación cruzada 7-fold con $Env + G$ el predictor	236
8.3	Ejemplo con variable respuesta categórica con optimización bayesiana con línea de partición aleatoria $Env + G + GE$ dentro del predictor	242
9	Partial Least Squares Regression.....	249
9.1	Ejemplo con respuesta continua solo G en el predictor con Grid search y particiones aleatorias	249
9.2	Ejemplo de datos de recueno con optimización bayesiana con línea de partición aleatoria $Env + G + GE$ en el predictor.	253
9.3	Ejemplo de multivariado con respuestas continuas con optimización bayesiana con validación cruzada 7-fold con $Env + G$ el predictor	258
9.4	Ejemplo con kernels con Grid search y particiones aleatorias	263
9.5	Eemplo con Sparse Kernels con Grid search y particiones aleatorias	271

1 Conjuntos de datos

Para ilustrar las funcionalidades de la biblioteca SKM con ejemplos, se mostrarán primero los conjuntos de datos *Chickpea* , *EYTTToy* , *GroundnutToy* y *MaizeToy* , que serán ilustrativos a lo largo de este manual. El siguiente código se puede usar para descargar todos esos conjuntos de datos en su directorio de trabajo actual:

```
base_url <- "https://gitlab.com/brandon-mosqueda/skm_toy_datasets/"

download.file(
  paste0(base_url, "ChickpeaToy.RData"),
  "ChickpeaToy.RData"
)
download.file(
  paste0(base_url, "EYTTToy.RData"),
  "EYTTToy.RData"
)
download.file(
  paste0(base_url, "GroundnutToy.RData"),
  "GroundnutToy.RData"
)
download.file(
  paste0(base_url, "MaizeToy.RData"),
  "MaizeToy.RData"
)
```

1.1 ChickpeaToy

El conjunto de datos de *ChickpeaToy* contiene dos data frames: *PhenoToy* y *GenoToy*.

```
# Import ChickpeaToy dataset
load("ChickpeaToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy
```

PhenoToy es un marco de datos de dimensión 180×8 que contiene las columnas *Line* , *Hyibrid_Name* y *Env* , además de cinco columnas de variables continuas: *Daysto50Flower* , *DaystoMaturity* , *AvePlantHeight* , *Biomass* y *PlantStand* .

Este conjunto de datos contiene 6 ambientes diferentes, con 30 líneas diferentes en cada uno.

```
# Dimension of PhenoToy
dim(PhenoToy)
#> [1] 180 8

# Columns of PhenoToy
colnames(PhenoToy)
#> [1] "Line" "Hybrid_Name" "Env" "Daysto50Flower"
```

```

#> [5] "DaystoMaturity" "AvePlantHeight" "Biomass"          "PlantStand"

# PhenoToy data summary
str(PhenoToy)
#> 'data.frame':    180 obs. of  8 variables:
#> $ Line          : Factor w/ 30 levels "ICCV00402","ICCV01301",...: 1 1 1 1 1 1 2 2 2 2 ...
#> $ Hybrid_Name    : Factor w/ 30 levels "ICCV00402","ICCV01301",...: 1 1 1 1 1 1 2 2 2 2 ...
#> $ Env            : Factor w/ 6 levels "1","2","4","5",...: 1 2 3 4 5 6 1 2 3 4 ...
#> $ Daysto50Flower: num  40.7 83 48.3 45 51 ...
#> $ DaystoMaturity: num  107 154 91.7 90.3 107.3 ...
#> $ AvePlantHeight: num  39.6 54.5 51 43.7 56.3 ...
#> $ Biomass        : num  235 392 249 359 310 ...
#> $ PlantStand     : num  11.7 35 11 11 12 ...

# First five elements of PhenoToy data
head(PhenoToy)
#>      Line Hybrid_Name Env Daysto50Flower DaystoMaturity AvePlantHeight
#> 1 ICCV00402 ICCV00402  1      40.66667      107.00000      39.56667
#> 2 ICCV00402 ICCV00402  2      83.00000      154.00000      54.50000
#> 3 ICCV00402 ICCV00402  4      48.33333      91.66667      51.00000
#> 4 ICCV00402 ICCV00402  5      45.00000      90.33333      43.66667
#> 5 ICCV00402 ICCV00402  6      51.00000     107.33333      56.33333
#> 6 ICCV00402 ICCV00402  7      53.33333      93.33333      38.00000
#>      Biomass PlantStand
#> 1 235.0000 11.66667
#> 2 392.5000 35.00000
#> 3 249.3333 11.00000
#> 4 358.6667 11.00000
#> 5 309.6667 12.00000
#> 6 198.0000 10.66667

# The number of individuals in each environment
table(PhenoToy$Env)
#>
#>  1  2  4  5  6  7
#> 30 30 30 30 30 30

# The number of unique lines
length(unique(PhenoToy$Line))
#> [1] 30

```

GenoToy es un data frame de dimensión 30×31 cuya primera columna es *Línea*, la cual contiene 30 líneas únicas (las mismas registradas en PhenoToy). El resto de las columnas son la información genotípica de cada línea.

```

# Dimension of GenoToy
dim(GenoToy)

```

```
#> [1] 30 31

# First rows and columns of GenoToy
head(GenoToy[, 1:5])
#>      Line ICCV00402 ICCV01301 ICCV03102 ICCV03104
#> ICCV00402 ICCV00402  2.393222  1.808427  1.657331  1.651927
#> ICCV01301 ICCV01301  1.808427  2.484758  1.685816  1.671517
#> ICCV03102 ICCV03102  1.657331  1.685816  2.257100  1.619275
#> ICCV03104 ICCV03104  1.651927  1.671517  1.619275  2.266445
#> ICCV03105 ICCV03105  1.740198  1.763279  1.705182  1.739184
#> ICCV03107 ICCV03107  1.718580  1.754947  1.654179  1.643145

# The number of unique lines
length(unique(GenoToy$Line))
#> [1] 30
```

Tenga en cuenta que, excluyendo la columna *Línea* , GenoToy es una matriz cuadrada.

1.2 EYTTToy

El conjunto de datos EYTTToy contiene dos data frames: PhenoToy y GenoToy.

```
# Import EYTTToy dataset
load("EYTTToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy
```

PhenoToy es un 120×7 data frame de dimensión que contiene las columnas *Line* , *Hybrid_Name* y *Env* , además de cuatro columnas de variables continuas: *DTHD* , *DTMT* , *GY* y *Height* .

Este conjunto de datos contiene 4 ambientes diferentes, con 30 líneas diferentes en cada uno.

```
# Dimension of PhenoToy
dim(PhenoToy)
#> [1] 120 7

# Columns of PhenoToy
colnames(PhenoToy)
#> [1] "Line"      "Hybrid_Name" "Env"          "DTHD"
#> [5] "DTMT"      "GY"          "Height"

# PhenoToy data summary
str(PhenoToy)
#> 'data.frame': 120 obs. of 7 variables:
#> $ Line : Factor w/ 30 levels "GID7462121","GID7625106",...: 1 1 1 1
#> 2 2 2 2 3 3 ...
#> $ Hybrid_Name: Factor w/ 30 levels "GID7462121","GID7625106",...: 1 1 1 1
```

```

2 2 2 2 3 3 ...
#> $ Env      : Factor w/ 4 levels "Bed5IR","EHT",...: 1 2 3 4 1 2 3 4 1 2
...
#> $ DTHD      : num  67 68 75 71 76 67 75 72 76 65 ...
#> $ DTMT      : num  109 123 109 106 114 127 114 107 113 125 ...
#> $ GY        : num  5.51 6.09 6.75 2.75 6.4 ...
#> $ Height    : num  107 100 103 98 91 90 98 88 98 83 ...

# First four elements of PhenoToy data
head(PhenoToy)
#>      Line Hybrid_Name      Env DTHD DTMT      GY Height
#> 1 GID7462121 GID7462121  Bed5IR   67  109 5.510785    107
#> 2 GID7462121 GID7462121   EHT    68  123 6.087132    100
#> 3 GID7462121 GID7462121 Flat5IR   75  109 6.754944    103
#> 4 GID7462121 GID7462121 FlatDrip  71  106 2.752278     98
#> 5 GID7625106 GID7625106  Bed5IR   76  114 6.399115     91
#> 6 GID7625106 GID7625106   EHT    67  127 5.951386     90

# The number of individuals in each environment
table(PhenoToy$Env)
#>
#>  Bed5IR      EHT  Flat5IR FlatDrip
#>      30      30      30      30

# The number of unique lines
length(unique(PhenoToy$Line))
#> [1] 30

```

GenoToy es un data frame de dimensión 30×31 cuya primera columna es *Línea* , la cual contiene 30 líneas únicas (las mismas registradas en PhenoToy). El resto de las columnas son la información genotípica de cada línea.

```

# Dimension of PhenoToy
dim(GenoToy)
#> [1] 30 31

# First rows and columns of GenoToy
head(GenoToy[, 1:5])
#>      Line GID7462121 GID7625106 GID7625276
#> GID7462121 GID7462121  0.91638818 -0.05329051 -0.17163954
#> GID7625106 GID7625106 -0.05329051  0.89803513  0.15992177
#> GID7625276 GID7625276 -0.17163954  0.15992177  0.99173614
#> GID7625985 GID7625985  0.06121540 -0.08227673  0.01615863
#> GID7626366 GID7626366 -0.03811005  0.03498484 -0.03254301
#> GID7626381 GID7626381 -0.09832617  0.02642334  0.05209018
#>      GID7625985
#> GID7462121  0.061215403
#> GID7625106 -0.082276733
#> GID7625276  0.016158626
#> GID7625985  0.934317080

```

```
#> GID7626366 0.009011423
#> GID7626381 -0.074658292

# The number of unique lines
length(unique(GenoToy$Line))
#> [1] 30
```

Tenga en cuenta que, excluyendo la columna *Línea*, *GenoToy* es una matriz cuadrada.

1.3 GroundnutToy

El conjunto de datos llamado *GroundnutToy* contiene dos marcos de datos: *PhenoToy* y *GenoToy*.

```
# Import GroundnutToy dataset
load("GroundnutToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy
```

PhenoToy es un 120×7 data frame que contiene las columnas *Line*, *Hybrid_Name* y *Env*, además de tres columnas de variables continuas (*NPP*, *PYPP* y *SYPP*) y una de conteo (*YPH*) que son variables de respuesta.

Este conjunto de datos contiene 4 ambientes diferentes, con 30 líneas diferentes en cada uno.

```
# Dimension of PhenoToy
dim(PhenoToy)
#> [1] 120 7

# Columns of PhenoToy
colnames(PhenoToy)
#> [1] "Line"          "Hybrid_Name" "Env"          "NPP"
#> [5] "PYPP"          "SYPP"        "YPH"

# PhenoToy data summary
str(PhenoToy)
#> 'data.frame': 120 obs. of 7 variables:
#> $ Line : Factor w/ 30 levels "49x37-134", "49x37-99(b)tall",...: 1 1
1 1 2 2 2 2 3 3 ...
#> $ Hybrid_Name: Factor w/ 30 levels "49x37-134", "49x37-99(b)tall",...: 1 1
1 1 2 2 2 2 3 3 ...
#> $ Env : Factor w/ 4 levels "ALIYARNAGAR_R15",...: 1 2 3 4 1 2 3 4 1
2 ...
#> $ NPP : num 19.05 8.25 12.96 13.6 10.2 ...
#> $ PYPP : num 13.45 6.04 7.18 7.6 12.29 ...
#> $ SYPP : num 7.23 3.97 3.74 4.03 6.7 2.07 3.97 8.68 4.69 1.5 ...
#> $ YPH : num 747 1614 1454 998 755 ...
```



```
# First five elements of PhenoToy data
head(PhenoToy)
#>           Line      Hybrid_Name      Env    NPP  PYPP
#> 1      49x37-134      49x37-134 ALIYARNAGAR_R15 19.05 13.45
#> 2      49x37-134      49x37-134 ICRISAT_PR15-16  8.25  6.04
#> 3      49x37-134      49x37-134      ICRISAT_R15 12.96  7.18
#> 4      49x37-134      49x37-134      JALGOAN_R15 13.60  7.60
#> 5 49x37-99(b)tall 49x37-99(b)tall ALIYARNAGAR_R15 10.20 12.29
#> 6 49x37-99(b)tall 49x37-99(b)tall ICRISAT_PR15-16  3.88  3.71
#>      SYPP      YPH
#> 1 7.23 746.90
#> 2 3.97 1614.19
#> 3 3.74 1454.29
#> 4 4.03  998.40
#> 5 6.70  754.60
#> 6 2.07  735.82

# The number of individuals in each environment
table(PhenoToy$Env)
#>
#> ALIYARNAGAR_R15 ICRISAT_PR15-16      ICRISAT_R15
#>              30              30              30
#>      JALGOAN_R15
#>              30

# The number of unique lines
length(unique(PhenoToy$Line))
#> [1] 30
```

GenoToy es un data frame de dimensión 30×31 cuya primera columna es *Línea* , la cual contiene 30 líneas únicas (las mismas registradas en PhenoToy). El resto de las columnas son la información genotípica de cada línea.

```
# Dimension of PhenoToy
dim(GenoToy)
#> [1] 30 31

# First rows and columns of GenoToy
head(GenoToy[, 1:4])
#>           Line 49x37-134 49x37-99(b)tall
#> 49x37-134      49x37-134 2.923524      2.603461
#> 49x37-99(b)tall 49x37-99(b)tall 2.603461      2.866530
#> CSMG84-1      CSMG84-1 2.223379      2.320184
#> DTG15      DTG15 2.703049      2.614714
#> DTG3      DTG3 2.659366      2.535092
#> Gangapuri      Gangapuri 2.612778      2.601767
#>      CSMG84-1
#> 49x37-134      2.223379
#> 49x37-99(b)tall 2.320184
#> CSMG84-1      2.288601
```

```
#> DTG15          2.221200
#> DTG3           2.150653
#> Gangapuri      2.238141

# The number of unique lines
length(unique(GenoToy$Line))
#> [1] 30
```

Tenga en cuenta que, excluyendo la columna *Línea* , GenoToy es una matriz cuadrada.

1.4 MaizeToy

El conjunto de datos de *MaizeToy* contiene dos data frames: PhenoToy y GenoToy.

```
# Import MaizeToy dataset
load("MaizeToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy
```

PhenoToy es un data frame de dimensión 90×6 que contiene las columnas *Line* , *Hybrid_Name* y *Env* , además de dos columnas de variables continuas *Yield* y *ASI* y una de conteo *PH* .

Este conjunto de datos contiene 3 ambientes diferentes, con 30 líneas diferentes en cada uno.

```
# Dimension of PhenoToy
dim(PhenoToy)
#> [1] 90 6

# Columns of PhenoToy
colnames(PhenoToy)
#> [1] "Line"          "Hybrid_Name" "Env"          "Yield"
#> [5] "ASI"           "PH"

# PhenoToy data summary
str(PhenoToy)
#> 'data.frame': 90 obs. of 6 variables:
#> $ Line : Factor w/ 30 levels "CKDHL0027","CKDHL0032",...: 1 1 1 2 2
2 3 3 3 4 ...
#> $ Hybrid_Name: Factor w/ 30 levels "CKDHL0027","CKDHL0032",...: 1 1 1 2 2
2 3 3 3 4 ...
#> $ Env : Factor w/ 3 levels "EBU","KAK","KTI": 1 2 3 1 2 3 1 2 3 1
...
#> $ Yield : num 6.11 6.21 5.32 6.62 5.6 6.24 5.24 4.93 6.7 4.72 ...
#> $ ASI : num 1.4 1 2 2 1.4 1.3 3 1.4 3.7 2.3 ...
#> $ PH : int 239 223 223 239 213 221 237 152 195 252 ...

# First four elements of PhenoToy data
```

```

head(PhenoToy)
#>      Line Hybrid_Name Env Yield ASI  PH
#> 1 CKDHL0027  CKDHL0027 EBU  6.11 1.4 239
#> 2 CKDHL0027  CKDHL0027 KAK  6.21 1.0 223
#> 3 CKDHL0027  CKDHL0027 KTI  5.32 2.0 223
#> 4 CKDHL0032  CKDHL0032 EBU  6.62 2.0 239
#> 5 CKDHL0032  CKDHL0032 KAK  5.60 1.4 213
#> 6 CKDHL0032  CKDHL0032 KTI  6.24 1.3 221

# The number of individuals in each environment
table(PhenoToy$Env)
#>
#> EBU KAK KTI
#> 30  30  30

# The number of unique lines
length(unique(PhenoToy$Line))
#> [1] 30

```

GenGenoToy es un data frame de dimensión cuya primera columna es *Línea*, la cual contiene 30 líneas únicas (las mismas registradas en PhenoToy).

```

# Dimension of GenoToy
dim(GenoToy)
#> [1] 30 31

# First rows and columns of GenoToy
head(GenoToy[, 1:5])
#>      Line  CKDHL0027  CKDHL0032  CKDHL0046
#> CKDHL0027 CKDHL0027 0.21293549 0.13298713 0.02787305
#> CKDHL0032 CKDHL0032 0.13298713 0.19495722 0.02753584
#> CKDHL0046 CKDHL0046 0.02787305 0.02753584 0.12543356
#> CKDHL0049 CKDHL0049 0.13768884 0.14035764 0.02908702
#> CKDHL0050 CKDHL0050 0.09827347 0.09791699 0.02945314
#> CKDHL0052 CKDHL0052 0.14445237 0.12033683 0.02830661
#>      CKDHL0049
#> CKDHL0027 0.13768884
#> CKDHL0032 0.14035764
#> CKDHL0046 0.02908702
#> CKDHL0049 0.21588369
#> CKDHL0050 0.13337251
#> CKDHL0052 0.11973948

# The number of unique lines
length(unique(GenoToy$Line))
#> [1] 30

```

GenoToy es un data frame de dimensión 30×31 cuya primera columna es *Línea*, la cual contiene 30 líneas únicas (las mismas registradas en PhenoToy). El resto de las columnas son la información genotípica de cada línea.

```
# Dimension of PhenoToy
dim(GenoToy)

# First rows and columns of GenoToy
head(GenoToy[, 1:5])

# The number of unique Lines
length(unique(GenoToy$Line))
```

Tenga en cuenta que, excluyendo la columna *Línea*, GenoToy es una matriz cuadrada.

2 Antes de empezar

2.1 Instalación SKM

Actualmente, SKM solo se puede instalar desde GitHub, puede usar el siguiente código para instalar la biblioteca desde allí:

```
if (!require(devtools)) {
  install.packages("devtools")
}

devtools::install_github("brandon-mosqueda/SKM")
```

Una vez instalado, debe importar la biblioteca con el siguiente código. Algunos ejemplos usan la biblioteca dplyr, por lo que se recomienda importarla también.

```
library(SKM)
library(dplyr)
```

2.2 Otras dependencias para los ejemplos

BurStMisc R se usa en algunos de los ejemplos para convertir algunas variables en variables binarias o categóricas, por lo que si no tiene instalada esta biblioteca en su máquina, puede usar el siguiente código para instalarla:

```
if (!require(BurStMisc)) {
  install.packages("BurStMisc")
}
```

3 Regresión generalizada

3.1 Ejemplo de resultados continuos solo G en el predictor con Grid search y particiones aleatorias

Este ejemplo evalúa un modelo lineal generalizado con cinco particiones aleatorias, con un 20 % de datos para el conjunto de prueba y un 80 % para el conjunto de entrenamiento

dentro de cada partición (los parámetros predeterminados de la función `cv_random`), para una respuesta continua, usando solo la matriz G (Matriz de diseño de línea que contiene información genómica) como predictor y utilizando "Grid_Search" como tipo de tuneo para el hiperparámetro α cuyas opciones son 0, 0.25, 0.50, 0.75 y 1.

En este ejemplo, el conjunto de datos utilizado es *GroundnutToy* y el objetivo es predecir la variable continua *YPH* del data frame *PhenoToy* utilizando la matriz G descrita anteriormente como predictor, por lo que identificamos las variables predictoras y de respuesta como X y y , respectivamente.

```
# Load the data
load("GroundnutToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of G
Line <- model.matrix(~ 0 + Line, data = PhenoToy)

# First column is Line
Geno <- cholesky(GenoToy[, -1])

# G matrix
LineG <- Line %%% Geno

# Predictor and Response Variables
X <- LineG
y <- PhenoToy$YPH

# Note that y is a continuous numeric vector
class(y)
#> [1] "numeric"
typeof(y)
#> [1] "double"
```

Tenga en cuenta que la variable de respuesta y es una variable continua (un vector con elementos de tipo "double"), lo cual es importante para que el modelo se entrene automáticamente para una variable continua.

Posteriormente, realizamos cinco particiones aleatorias, con 80 los datos para el conjunto de entrenamiento y 20 para el conjunto de prueba, con la ayuda de la función `cv_random` (con los parámetros por defecto). Además, creamos los data frames vacíos *Predictionse* *Hyperparams* que servirán para guardar los valores observados y predichos en cada entorno y posteriormente evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
# Set seed for reproducible results
set.seed(2022)
```

```
# Random Partition
folds <- cv_random(records_number = nrow(X))

# A data frame that will contain the variables:
# (Number) Fold, Line, Env, (testing values)
# Observed and Predicted (values)
Predictions <- data.frame()
Hyperparams <- data.frame()
```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta.
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo los valores 0, 0.25, 0.50, 0.75 y 1 para el hiperparámetro α y 100 como el número de lambdas generadas para tunear los hiperparámetros (parámetro por defecto de *lambdas_number*) con "Grid Search" como tipo de tuneo (parámetro por defecto de *tune_type*).
3. Con el modelo obtenido en (2), se predice la variable respuesta *YPH* en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba.
 - b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*.
5. Identificación de hiperparámetros:
 - a. *HyperparamsFold*, que contiene los valores *alpha* del modelo obtenido en (2), *loss* que es el costo del modelo para cada hiperparámetro α y el número de *Fold*.
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
6. Se muestran los hiperparámetros óptimos.

```
# Model training and predictions of the ith partition
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")
  fold <- folds[[i]]

  # Identify the training and testing sets
  X_training <- X[fold$training, ]
  X_testing <- X[fold$testing, ]
  y_training <- y[fold$training]
  y_testing <- y[fold$testing]

  # Model training
  model <- generalized_linear_model(
```

```

x = X_training,
y = y_training,

# Specify the hyperparameters
alpha = c(0, 0.25, 0.50, 0.75, 1),
lambdas_number = 100,
tune_type = "grid_search"
)

# Prediction of the test set
predictions <- predict(model, X_testing)

# Predictions for the Fold
FoldPredictions <- data.frame(
  Fold = i,
  Line = PhenoToy$Line[fold$testing],
  Env = PhenoToy$Env[fold$testing],
  Observed = y_testing,
  Predicted = predictions$predicted
)
Predictions <- rbind(Predictions, FoldPredictions)

# Hyperparams for the Fold
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)

# Best hyperparams of the model
cat("*** Optimal hyperparameters: ***\n")
print(model$best_hyperparams)
}
#> *** Fold: 1 ***
#> *** Grid Search Tuning ***
#> Total combinations: 5
#> Combination: 1 / 5
#> Combination: 2 / 5
#> Combination: 3 / 5
#> Combination: 4 / 5
#> Combination: 5 / 5
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 0.3932 secs ***
#> *** Optimal hyperparameters: ***
#> $alpha
#> [1] 0
#>
#> $loss
#> [1] 352092.4
#>
#> $Lambda

```

```

#> [1] 134.508
#>
#> *** Fold: 2 ***
#> *** Grid Search Tuning ***
#> Total combinations: 5
#> Combination: 1 / 5
#> Combination: 2 / 5
#> Combination: 3 / 5
#> Combination: 4 / 5
#> Combination: 5 / 5
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 0.1772 secs ***
#> *** Optimal hyperparameters: ***
#> $alpha
#> [1] 0
#>
#> $loss
#> [1] 381594.2
#>
#> $lambda
#> [1] 75.30319
#>
#> *** Fold: 3 ***
#> *** Grid Search Tuning ***
#> Total combinations: 5
#> Combination: 1 / 5
#> Combination: 2 / 5
#> Combination: 3 / 5
#> Combination: 4 / 5
#> Combination: 5 / 5
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 0.1441 secs ***
#> *** Optimal hyperparameters: ***
#> $alpha
#> [1] 0.25
#>
#> $loss
#> [1] 421160.7
#>
#> $lambda
#> [1] 1288.189
#>
#> *** Fold: 4 ***
#> *** Grid Search Tuning ***
#> Total combinations: 5
#> Combination: 1 / 5
#> Combination: 2 / 5
#> Combination: 3 / 5
#> Combination: 4 / 5
#> Combination: 5 / 5

```



```

#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 0.1661 secs ***
#> *** Optimal hyperparameters: ***
#> $alpha
#> [1] 0.25
#>
#> $loss
#> [1] 600870.4
#>
#> $lambda
#> [1] 263.8843
#>
#> *** Fold: 5 ***
#> *** Grid Search Tuning ***
#> Total combinations: 5
#> Combination: 1 / 5
#> Combination: 2 / 5
#> Combination: 3 / 5
#> Combination: 4 / 5
#> Combination: 5 / 5
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 0.2064 secs ***
#> *** Optimal hyperparameters: ***
#> $alpha
#> [1] 1
#>
#> $loss
#> [1] 386449.7
#>
#> $lambda
#> [1] 316.2567

```

Predictions contiene las columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo el hiperparámetro óptimo (entre los valores posibles de α) que minimiza la función de costo con el tipo de tuneo "Grid Search", correspondiente al formato necesario para utilizar la función *gs_summaries* sobre estas predicciones en el caso de variables continuas.

La función *gs_summaries* devuelve una lista que contiene tres marcos de datos correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```

head(Predictions)
#>   Fold      Line      Env Observed Predicted
#> 1    1 ICGV97115  JALGOAN_R15    817.85  1750.316
#> 2    1  ICG9315   ICRISAT_R15   1324.07  1351.960
#> 3    1 ICGV06099 ICRISAT_PR15-16  2334.15  2337.382
#> 4    1 ICGV00248   ICRISAT_R15   1993.36  1650.276
#> 5    1 ICGV05057   ICRISAT_R15   1856.64  1761.282
#> 6    1 ICGV02434  JALGOAN_R15    367.32  1654.256

```

```

unique(Predictions$Fold)
#> [1] 1 2 3 4 5
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"

# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted Difference
#> 1  ICG15419 1307.694  1297.380    10.3143
#> 2  Gangapuri 1232.337  1266.256    33.9193
#> 3   ICG9315 1453.340  1372.802    80.5380
#> 4   ICG3746 1257.120  1172.744    84.3762
#> 5 ICGV07217 1530.160  1427.290   102.8700
#> 6      TG19 1106.877  1225.080   118.2030

# Summaries by Environment
summaries$env[, 1:7]
#>      Env      MSE    MSE_SE    RMSE  RMSE_SE  NRMSE
#> 1 ALIYARNAGAR_R15 233853.8  94263.47 433.1082 107.5535 1.4550
#> 2 ICRISAT_PR15-16 474541.6 215639.36 632.3581 136.6243 1.1490
#> 3   ICRISAT_R15 194575.4  39698.08 429.7547  49.7150 0.7642
#> 4   JALGOAN_R15 778332.8 213434.09 843.6534 129.0172 0.9526
#> 5      Global 436937.2 125174.49 635.1584  91.5301 0.8571
#>      NRMSE_SE
#> 1    0.9269
#> 2    0.1384
#> 3    0.0864
#> 4    0.0910
#> 5    0.0323
summaries$env[, 8:14]
#>      MAE    MAE_SE    Cor Cor_SE  Intercept Intercept_SE
#> 1 368.6256  96.0409 0.8289 0.0977   -674.4813    571.6053
#> 2 487.1560 104.2707 0.3282 0.1236  -1060.0183   2231.7069
#> 3 369.9474  44.8257 0.6574 0.0847   -601.0640    385.9712
#> 4 679.6807  93.6651 0.2878 0.3075   -593.6681   1728.2567
#> 5 481.8010  54.2532 0.5876 0.0710   -792.6003    867.6839
#>      Slope
#> 1 1.3417
#> 2 2.0167
#> 3 1.4151
#> 4 1.4387
#> 5 1.6100
summaries$env[, 15:19]
#>      Slope_SE    R2    R2_SE  MAAPE  MAAPE_SE
#> 1    0.4659 0.7253 0.1365 0.3336    0.1022
#> 2    1.7325 0.1688 0.0853 0.3053    0.0477
#> 3    0.3240 0.4609 0.1261 0.2596    0.0379

```

```
#> 4 1.2318 0.4610 0.1403 0.3801 0.0583
#> 5 0.6433 0.3655 0.0793 0.3004 0.0118

# Summaries by Fold
summaries$fold[, 1:8]
#>      Fold      MSE      MSE_SE      RMSE      RMSE_SE      NRMSE      NRMSE_SE
#> 1      1 570690.9 284881.87 672.5285 198.6591 0.8023 0.1584
#> 2      2 390728.2 209952.64 543.4464 178.3201 0.6942 0.1915
#> 3      3 357208.0 69021.73 590.0753 54.8306 2.1112 1.0276
#> 4      4 187967.8 59984.46 417.7342 66.9975 0.8433 0.1582
#> 5      5 595034.5 300472.71 699.8087 187.3520 0.9499 0.0466
#> 6 Global 436937.2 125174.49 635.1584 91.5301 0.8571 0.0323
#>      MAE
#> 1 544.8338
#> 2 426.9615
#> 3 495.7907
#> 4 362.8980
#> 5 551.2781
#> 6 481.8010
```

Además, Hyperparams contiene las columnas *alpha*, *loss* y *Fold*, donde el valor de la columna de pérdida corresponde al costo del modelo para cada combinación de los α valores de partición y, ordenados de menor a mayor dentro de cada partición.

```
head(Hyperparams)
#>   alpha      Loss Fold
#> 1 0.00 352092.4     1
#> 2 0.25 365345.9     1
#> 3 0.50 372066.7     1
#> 4 0.75 373700.4     1
#> 5 1.00 374623.0     1
#> 6 0.00 381594.2     2

# Last rows of Hyperparams
tail(Hyperparams)
#>   alpha      Loss Fold
#> 42 0.75 605829.9     4
#> 53 1.00 386449.7     5
#> 43 0.75 387226.6     5
#> 33 0.50 387556.5     5
#> 23 0.25 387943.8     5
#> 13 0.00 388567.6     5
```

3.2 Ejemplo con respuesta binaria con Grid search y validación cruzada 7-fold con *env* + *g* en el predictor.

Este ejemplo evalúa un modelo lineal generalizado con validación cruzada 7-fold, para una respuesta binaria, utilizando el efecto ambiente y la matriz *G* como predictores, además de

"Grid Search" como tipo de tuneo para el hiperparámetro α , cuyas opciones son 0, 0.25, 0.50, 0.75 y 1.

En este ejemplo, el conjunto de datos utilizado es *ChickpeaToy* y el objetivo es predecir la variable binaria y_{bin} , que es una transformación de la variable *Biomass* que indica si la respuesta es mayor que la mediana de esta variable o no, utilizando la matriz de diseño de la variable *Env de PhenoToy* y matrix *G*, descritos anteriormente, como predictores; por lo que identificamos las variables predictoras y de respuesta como *X* y y_{bin} en respectivamente. Tenga en cuenta que para que la variable se reconozca como una variable binaria al entrenar el modelo y luego usar *gs_summaries*, la variable de respuesta debe ser un factor.

```
# Load the data
load("ChickpeaToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env & G
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %*% Geno

# Predictor and Response Variables
X <- cbind(Env, LineG)

# Binary response as factor
y_bin <- BurStMisc::ntile(PhenoToy$Biomass, 2, result = "factor")

# First 30 responses
print(y_bin[1:30])
#> [1] 1 2 1 2 2 1 1 1 1 1 2 1 2 2 1 2 2 2 2 2 1 1 2 1 2 2
#> [30] 1
#> Levels: 1 < 2
```

Tenga en cuenta que la variable de respuesta y_{bin} es un factor con solo dos niveles (o categorías), lo cual es importante para que el modelo se entrene automáticamente para una variable binaria (regresión logística). Por esta razón, es importante tener en cuenta esas variables de respuesta binarias o categóricas antes de usar la función *generalized_linear_model*.

Posteriormente hacemos las particiones correspondientes a 7-Fold CV, con la ayuda de la función *cv_kfold*. Además, creamos los data frames vacíos *Predictions* e *Hyperparams* que servirán para guardar los valores observados y predichos en cada ambiente y posteriormente evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
# Set seed for reproducible results
set.seed(2022)
# Generates the folds
folds <- cv_kfold(records_number = nrow(X), k = 7)

# A data frame that will contain the variables:
# (Number) Fold, Line, Env, (testing values) Observed,
# Predicted (values) and the predicted probabilities of
# responses
Predictions <- data.frame()
Hyperparams <- data.frame()
```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo los valores 0, 0.25, 0.50, 0.75 y 1 para el hiperparámetro α y 100 como el número de lambdas generadas para tunear los hiperparámetros (parámetro por defecto de *lambdas_number*) con "Grid Search" como tipo de tuneo (parámetro por defecto de *tune_type*);
3. Con el modelo obtenido en (2), y_{bin} se predice la variable respuesta en el conjunto de prueba, con el fin de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: *Fold* number, *Line*, *Env*, *Observed*, *Predicted*, 1 y 2 para cada elemento del conjunto de prueba. Tenga en cuenta que, a diferencia del ejemplo anterior, ahora tenemos dos columnas adicionales correspondientes a las probabilidades asociadas con cada elemento correspondiente a esa categoría.
 - b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*. Con esto, *Predictions* se formatea para que sea un argumento de la función *gs_summaries*.
5. Identificación de hiperparámetros:
 - a. *HyperparamsFold*, que contiene los valores *alpha* del modelo obtenido en (2), *loss*, que es el costo del modelo para cada hiperparámetro α y el número de *Fold*.
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
6. Se muestran los hiperparámetros óptimos.

```
# Model training and predictions of the ith partition
for (i in seq_along(folds)) {
```

```

cat("*** Fold:", i, "***\n")
fold <- folds[[i]]

# Identify the training and testing sets
X_training <- X[fold$training, ]
X_testing <- X[fold$testing, ]
y_training <- y_bin[fold$training]
y_testing <- y_bin[fold$testing]

# Model training
model <- generalized_linear_model(
  x = X_training,
  y = y_training,

  # Specify the hyperparameter ranges
  alpha = c(0, 0.25, 0.50, 0.75, 1),
  lambdas_number = 100,
  tune_type = "grid_search",
)

# Testing Predictions
predictions <- predict(model, X_testing)

# Predictions for the Fold Fold
FoldPredictions <- cbind(
  data.frame(
    Fold = i,
    Line = PhenoToy$Line[fold$testing],
    Env = PhenoToy$Env[fold$testing],
    Observed = y_testing,
    Predicted = predictions$predicted
  ),
  predictions$probabilities
)
Predictions <- rbind(Predictions, FoldPredictions)

# Hyperparams for the Fold
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)

# Best hyperparams of the model
cat("*** Optimal hyperparameters: ***\n")
print(model$best_hyperparams)
}
#> *** Fold: 1 ***
#> *** Grid Search Tuning ***
#> Total combinations: 5
#> Combination: 1 / 5

```

```

#> Combination: 2 / 5
#> Combination: 3 / 5
#> Combination: 4 / 5
#> Combination: 5 / 5
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 0.3206 secs ***
#> *** Optimal hyperparameters: ***
#> $alpha
#> [1] 0.25
#>
#> $loss
#> [1] 0.2077922
#>
#> $lambda
#> [1] 0.0142669
#>
#> *** Fold: 2 ***
#> *** Grid Search Tuning ***
#> Total combinations: 5
#> Combination: 1 / 5
#> Combination: 2 / 5
#> Combination: 3 / 5
#> Combination: 4 / 5
#> Combination: 5 / 5
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 0.3082 secs ***
#> *** Optimal hyperparameters: ***
#> $alpha
#> [1] 0.75
#>
#> $loss
#> [1] 0.2012987
#>
#> $lambda
#> [1] 0.03560764
#>
#> *** Fold: 3 ***
#> *** Grid Search Tuning ***
#> Total combinations: 5
#> Combination: 1 / 5
#> Combination: 2 / 5
#> Combination: 3 / 5
#> Combination: 4 / 5
#> Combination: 5 / 5
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 0.2881 secs ***
#> *** Optimal hyperparameters: ***
#> $alpha
#> [1] 0
#>

```

```

#> $loss
#> [1] 0.2322581
#>
#> $lambda
#> [1] 0.04496679
#>
#> *** Fold: 4 ***
#> *** Grid Search Tuning ***
#> Total combinations: 5
#> Combination: 1 / 5
#> Combination: 2 / 5
#> Combination: 3 / 5
#> Combination: 4 / 5
#> Combination: 5 / 5
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 0.3341 secs ***
#> *** Optimal hyperparameters: ***
#> $alpha
#> [1] 0.25
#>
#> $loss
#> [1] 0.1688312
#>
#> $lambda
#> [1] 0.04285206
#>
#> *** Fold: 5 ***
#> *** Grid Search Tuning ***
#> Total combinations: 5
#> Combination: 1 / 5
#> Combination: 2 / 5
#> Combination: 3 / 5
#> Combination: 4 / 5
#> Combination: 5 / 5
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 0.3536 secs ***
#> *** Optimal hyperparameters: ***
#> $alpha
#> [1] 0.25
#>
#> $loss
#> [1] 0.1935484
#>
#> $lambda
#> [1] 0.04945194
#>
#> *** Fold: 6 ***
#> *** Grid Search Tuning ***
#> Total combinations: 5
#> Combination: 1 / 5

```



```

#> Combination: 2 / 5
#> Combination: 3 / 5
#> Combination: 4 / 5
#> Combination: 5 / 5
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 0.3073 secs ***
#> *** Optimal hyperparameters: ***
#> $alpha
#> [1] 0.25
#>
#> $loss
#> [1] 0.2012987
#>
#> $lambda
#> [1] 0.3815682
#>
#> *** Fold: 7 ***
#> *** Grid Search Tuning ***
#> Total combinations: 5
#> Combination: 1 / 5
#> Combination: 2 / 5
#> Combination: 3 / 5
#> Combination: 4 / 5
#> Combination: 5 / 5
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 0.2902 secs ***
#> *** Optimal hyperparameters: ***
#> $alpha
#> [1] 1
#>
#> $loss
#> [1] 0.2012987
#>
#> $lambda
#> [1] 0.02228067

```

Predictions contienen las columnas *Fold*, *Line*, *Env*, *Observed*, *Predicted*, 1 y 2 para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo el hiperparámetro óptimo (entre los valores posibles de α) que minimiza el costo con el tipo de tuneo "Grid Search", correspondiente al formato necesario para utilizar la función *gs_summaries* sobre *Prediction* en el caso de variables binarias.

La función *gs_summaries* devuelve una lista que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```

head(Predictions)
#>   Fold      Line Env Observed Predicted      1      2
#> 4     1 ICCV00402  5         2         2 0.35047563 0.64952437
#> 6     1 ICCV00402  7         1         1 0.98387196 0.01612804
#> 7     1 ICCV01301  1         1         1 0.93715978 0.06284022

```

```

#> 10      1 ICCV01301      5      1      1 0.69555369 0.30444631
#> 17      1 ICCV03102      6      2      2 0.01205161 0.98794839
#> 21      1 ICCV03104      4      2      2 0.39746731 0.60253269
unique(Predictions$Fold)
#> [1] 1 2 3 4 5 6 7
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"

# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted      X1      X2
#> 1 ICCV00402          1          2 0.4620 0.5380
#> 2 ICCV01301          1          1 0.6677 0.3323
#> 3 ICCV03102          2          1 0.4540 0.5460
#> 4 ICCV03104          2          2 0.2851 0.7149
#> 5 ICCV03105          1          2 0.4206 0.5794
#> 6 ICCV03107          2          2 0.4569 0.5431

# Summaries by Environment
summaries$env
#>      Env      PCCC PCCC_SE      Kappa Kappa_SE BrierScore
#> 1      1 0.5163 0.1322 0.0141 0.0782 0.6215
#> 2      2 0.6857 0.1523 -0.1607 0.0892 0.3630
#> 3      4 0.7738 0.0673 0.1667 0.1543 0.3520
#> 4      5 0.5952 0.1085 0.0538 0.1986 0.4852
#> 5      6 0.9714 0.0286 0.0000      NA 0.0998
#> 6      7 1.0000 0.0000      NaN      NA 0.0519
#> 7 Global 0.7414 0.0485 0.3988 0.1454 0.3590
#>      BrierScore_SE
#> 1      0.1793
#> 2      0.1405
#> 3      0.0534
#> 4      0.0665
#> 5      0.0494
#> 6      0.0223
#> 7      0.0415

# Summaries by Fold
summaries$fold
#>      FoId      PCCC PCCC_SE      Kappa Kappa_SE BrierScore
#> 1      1 0.9000 0.0683 0.4286 0.2103 0.1870
#> 2      2 0.8694 0.0609 -0.0476 0.0337 0.2561
#> 3      3 0.8417 0.0821 0.0707 0.1811 0.2249
#> 4      4 0.6167 0.1833 0.0000 0.0000 0.4685
#> 5      5 0.7250 0.1515 0.1364 0.1113 0.4588
#> 6      6 0.5857 0.1546 -0.2400 0.2094 0.4385

```

```
#> 7      7 0.7611 0.1124 0.0392 0.0277 0.2685
#> 8 Global 0.7414 0.0485 0.3988 0.1454 0.3590
#> BrierScore_SE
#> 1      0.0959
#> 2      0.0771
#> 3      0.0486
#> 4      0.1743
#> 5      0.2441
#> 6      0.0792
#> 7      0.1082
#> 8      0.0415
```

Además, Hyperpaams contiene las columnas *alpha*, *loss* y *Fold*, donde el valor de la columna de pérdida corresponde al costo del modelo para cada combinación de los α valores de partición y, ordenados de menor a mayor dentro de cada partición.

```
# First rows of Hyperparams
head(Hyperparams)
#>   alpha      Loss Fold
#> 2  0.25 0.2077922    1
#> 4  0.75 0.2077922    1
#> 5  1.00 0.2077922    1
#> 3  0.50 0.2077922    1
#> 1  0.00 0.2337662    1
#> 41 0.75 0.2012987    2

# Last rows of Hyperparams
tail(Hyperparams)
#>   alpha      Loss Fold
#> 15 0.00 0.2207792    6
#> 56 1.00 0.2012987    7
#> 16 0.00 0.2077922    7
#> 26 0.25 0.2077922    7
#> 46 0.75 0.2077922    7
#> 36 0.50 0.2077922    7
```

3.3 Ejemplo con variable respuesta categórica con optimización bayesiana con partición aleatoria *Env* + *G* + *GE* dentro del predictor

Este ejemplo evalúa un modelo lineal generalizado con cinco particiones aleatorias, con 20% los datos para el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de cada partición (los parámetros predeterminados de la función `cv_random`), para una respuesta categórica, usando el efecto del Ambiente, el matriz *G* y la interacción entre estos dos como predictores, además de utilizar "Bayesian_optimization" como un tipo de tuneo para el hiperparámetro α .

En este ejemplo, el conjunto de datos utilizado es *EYTT* y buscamos predecir la variable categórica *y*, que es una transformación de la variable *GY* del marco de datos de *PhenoToy* usando la función *ntile*, utilizando la matriz de diseño de *PhenoToy*. *Env* variable, la matriz

G descrita anteriormente y la matriz de diseño de la interacción entre estas dos, como predictores; por lo que identificamos las variables predictoras y de respuesta como X y y respectivamente.

```
# Load the dataset
load("EYTTToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env, G & GE
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)

# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %*% Geno
LineXGenoEnv <- model.matrix(~ 0 + LineG:Env)

# Predictor and Response Variables
X <- cbind(Env, LineG, LineXGenoEnv)
y <- BurStMisc::ntile(PhenoToy$GY, 3, result = "factor")

# First 30 responses
print(y[1:30])
#> [1] 1 2 3 1 2 2 2 1 3 2 2 1 2 2 3 1 3 3 3 1 3 3 3 1 2 3 3 1 3
#> [30] 3
#> Levels: 1 < 2 < 3
```

Tenga en cuenta que la variable de respuesta y es un factor con tres niveles (o categorías), lo cual es importante para que el modelo se entrene automáticamente para una variable categórica (**modelo multinomial simétrico**). Por esta razón, es importante tener en cuenta esas variables de respuesta binarias o categóricas antes de usar la función `generalized_linear_model`.

Posteriormente, realizamos cinco particiones aleatorias, con 80% los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función `cv_random` (con los parámetros por defecto). Además, creamos los data frames vacíos `Predictions` e `Hyperparams` que servirán para guardar los valores observados y predichos en cada ambiente y posteriormente evaluar la capacidad predictiva del modelo con la función `gs_summaries`.

```
# Set seed for reproducible results
set.seed(2022)
# Generate folds
folds <- cv_random(records_number = nrow(X))

# A data frame that will contain the variables:
```

```
Predictions <- data.frame()
Hyperparams <- data.frame()
```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo valores entre 0 y 1 para el hiperparámetro α y 100 como el número de lambdas generadas para ajustar los hiperparámetros (parámetro predeterminado de *lambdas_number*) con "Bayesian_optimization" como tipo de tuneo;
3. Con el modelo obtenido en (2), se predice la variable respuesta y en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed*, *Predicted*, 1, 2 y 3 para cada elemento del conjunto de prueba. Tenga en cuenta que, a diferencia de los ejemplos anteriores, ahora tenemos tres columnas adicionales correspondientes a las probabilidades asociadas con cada elemento correspondiente a esa categoría.
 - b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*. Con esto, *Predictions* se formatea para que sea un argumento de la función *gs_summaries*.
5. Identificación de hiperparámetros:
 - a. *HyperparamsFold* que contiene los valores *alpha* del modelo obtenido en (2), *precision*, que es la precisión del modelo para cada hiperparámetro α , y el número *Fold*.
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
6. Se muestran los hiperparámetros óptimos.

```
# Model training and predictions of the ith partition
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")
  fold <- folds[[i]]

  # Identify the training and testing sets
  X_training <- X[fold$training, ]
  X_testing <- X[fold$testing, ]
  y_training <- y[fold$training]
  y_testing <- y[fold$testing]

  # Model training
  model <- generalized_linear_model(
    x = X_training,
```

```

y = y_training,

# Specify the hyperparameter ranges
alpha = list(min = 0, max = 1),
lambdas_number = 100,
tune_type = "Bayesian_optimization",
tune_bayes_samples_number = 5,
tune_bayes_iterations_number = 5
)

# Testing Predictions
predictions <- predict(model, X_testing)

# Predictions for the Fold
FoldPredictions <- cbind(
  data.frame(
    Fold = i,
    Line = PhenoToy$Line[fold$testing],
    Env = PhenoToy$Env[fold$testing],
    Observed = y_testing,
    Predicted = predictions$predicted
  ),
  predictions$probabilities
)
Predictions <- rbind(Predictions, FoldPredictions)

# Hyperparams for the Fold
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)

# Best hyperparams of the model
cat("*** Optimal hyperparameters: ***\n")
print(model$best_hyperparams)
}
#> *** Fold: 1 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1
#> Combination: 2
#> Combination: 3
#> Combination: 4
#> Combination: 5
#> Combination: 6
#> Combination: 7
#> Combination: 8
#> Combination: 9
#> Combination: 10
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 6.6865 secs ***

```

```

#> *** Optimal hyperparameters: ***
#> $alpha
#> [1] 0.8490901
#>
#> $accuracy
#> [1] -0.2916667
#>
#> $lambda
#> [1] 0.406885
#>
#> *** Fold: 2 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1
#> Combination: 2
#> Combination: 3
#> Combination: 4
#> Combination: 5
#> Combination: 6
#> Combination: 7
#> Combination: 8
#> Combination: 9
#> Combination: 10
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 6.7918 secs ***
#> *** Optimal hyperparameters: ***
#> $alpha
#> [1] 0.730056
#>
#> $accuracy
#> [1] -0.3333333
#>
#> $lambda
#> [1] 0.1153609
#>
#> *** Fold: 3 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1
#> Combination: 2
#> Combination: 3
#> Combination: 4
#> Combination: 5
#> Combination: 6
#> Combination: 7
#> Combination: 8
#> Combination: 9
#> Combination: 10
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 6.0102 secs ***
#> *** Optimal hyperparameters: ***
#> $alpha

```

```

#> [1] 0.8376429
#>
#> $accuracy
#> [1] -0.2604167
#>
#> $lambda
#> [1] 0.3945454
#>
#> *** Fold: 4 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1
#> Combination: 2
#> Combination: 3
#> Combination: 4
#> Combination: 5
#> Combination: 6
#> Combination: 7
#> Combination: 8
#> Combination: 9
#> Combination: 10
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 5.8977 secs ***
#> *** Optimal hyperparameters: ***
#> $alpha
#> [1] 1
#>
#> $accuracy
#> [1] -0.3333333
#>
#> $lambda
#> [1] 0.1451336
#>
#> *** Fold: 5 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1
#> Combination: 2
#> Combination: 3
#> Combination: 4
#> Combination: 5
#> Combination: 6
#> Combination: 7
#> Combination: 8
#> Combination: 9
#> Combination: 10
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 6.0448 secs ***
#> *** Optimal hyperparameters: ***
#> $alpha
#> [1] 0.9287305
#>

```



```
#> $accuracy
#> [1] -0.3229167
#>
#> $lambda
#> [1] 0.1118448
```

Predictions contiene las columnas *Fold*, *Line*, *Env*, *Observed*, *Predicted*, 1, 2 y 3 para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo el hiperparámetro óptimo (entre los valores posibles de α) que minimiza la función de costo (*pcic* : Proporción de Casos Incorrectamente Clasificados) con el tipo de tuneo "Bayesian_optimization", correspondiente al formato necesario para usar la función *gs_summaries* sobre *Predicción* en el caso de variables categóricas.

La función *gs_summaries* devuelve una lista que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```
head(Predictions)
#>      Fold      Line      Env Observed Predicted      1
#> 100      1 GID7632666 FlatDrip      1      1 0.5505770
#> 51      1 GID7628158 Flat5IR      2      2 0.2644307
#> 78      1 GID7631195      EHT      3      3 0.2592754
#> 55      1 GID7628467 Flat5IR      3      3 0.2520352
#> 75      1 GID7630553 Flat5IR      3      3 0.2489198
#> 68      1 GID7629600 FlatDrip      1      1 0.5507031
#>           2           3
#> 100 0.2242722 0.2251507
#> 51  0.3697049 0.3658644
#> 78  0.3607363 0.3799882
#> 55  0.3523746 0.3955902
#> 75  0.3480189 0.4030613
#> 68  0.2246144 0.2246825

unique(Predictions$Fold)
#> [1] 1 2 3 4 5
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted      X1      X2      X3
#> 1 GID7462121      1      2 0.2191 0.3956 0.3853
#> 2 GID7625106      2      2 0.2310 0.3942 0.3748
#> 3 GID7625276      1      1 0.4168 0.3141 0.2691
#> 4 GID7625985      1      1 0.4526 0.2840 0.2634
#> 5 GID7626366      1      1 0.4345 0.1685 0.3970
#> 6 GID7626381      3      3 0.3037 0.1806 0.5157

# Summaries by Environment
```

```

summaries$env
#>      Env  PCCC PCCC_SE  Kappa Kappa_SE BrierScore
#> 1  Bed5IR 0.5905  0.0683 0.2594  0.0983    0.6494
#> 2    EHT 0.4733  0.0799 0.1738  0.1007    0.6720
#> 3 Flat5IR 0.5079  0.0528 0.0100  0.1222    0.5622
#> 4 FlatDrip 1.0000  0.0000  NaN      NA    0.1722
#> 5  Global 0.6685  0.0348 0.4956  0.0451    0.5216
#> BrierScore_SE
#> 1      0.0781
#> 2      0.0399
#> 3      0.0123
#> 4      0.0452
#> 5      0.0306

# Summaries by Fold
summaries$fold
#>      Fold  PCCC PCCC_SE  Kappa Kappa_SE BrierScore
#> 1      1 0.6012  0.1715  0.1380  0.1279    0.5457
#> 2      2 0.6528  0.1250 -0.0119  0.1340    0.4744
#> 3      3 0.5905  0.1472  0.1363  0.0948    0.5751
#> 4      4 0.6667  0.1247  0.2484  0.1742    0.4980
#> 5      5 0.7036  0.1081  0.2280  0.1320    0.4766
#> 6 Global 0.6685  0.0348  0.4956  0.0451    0.5216
#> BrierScore_SE
#> 1      0.0815
#> 2      0.1544
#> 3      0.1263
#> 4      0.1311
#> 5      0.1339
#> 6      0.0306

```

Además, Hyperparams contiene columnas *alpha*, *accuracy* y *Fold*, donde el valor de *accuracy* corresponde al costo del modelo para cada combinación de α los valores de partición y, ordenados de mayor a menor dentro de cada partición.

```

# First rows of Hyperparams
head(Hyperparams)
#>      alpha  accuracy Fold
#> 3 0.8490901 -0.2916667  1
#> 6 1.0000000 -0.2916667  1
#> 7 0.9289922 -0.3020833  1
#> 8 0.7743241 -0.3020833  1
#> 1 0.6804129 -0.3229167  1
#> 9 0.5467352 -0.3229167  1

# Last rows of Hyperparams
tail(Hyperparams)
#>      alpha  accuracy Fold
#> 104 1.0000000 -0.3229167  5
#> 14  0.41268630 -0.3333333  5

```

```
#> 44  0.62443722 -0.3333333  5
#> 74  0.27323122 -0.3437500  5
#> 34  0.14337780 -0.3541667  5
#> 24  0.05022715 -0.4166667  5
```

3.4 Ejemplo con datos de conteo con optimización bayesiana con línea de partición aleatoria *Env* + *G* + *GE* en el predictor

This example evaluates Este ejemplo evalúa un modelo lineal generalizado con cinco particiones aleatorias del conjunto de líneas, con 20% las líneas para el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de cada partición, para una respuesta de conteo, utilizando el efecto del ambiente, la matriz *G* y la interacción entre estos dos como predictores, además de usar la "Bayesian_optimization" como un tipo de tuneo para el hiperparámetro α .

En este ejemplo, el dataset utilizado es *MaizeToy* y busca predecir la variable de conteo numérico *PH*, utilizando como predictores la matriz de diseño de la variable *Env* de PhenoToy, la matriz *G* descrita anteriormente y la matriz de diseño de la interacción entre estas dos; por lo que identificamos las variables predictoras y de respuesta como *X* y *y* respectivamente.

```
# Load the dataset
load("MaizeToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env, G & GE
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
LineG <- Line %*% Geno
LineXGenoEnv <- model.matrix(~ 0 + LineG:Env)

# Predictor and Response Variables
X <- cbind(Env, LineG, LineXGenoEnv)
y <- PhenoToy$PH
print(y[1:15])
#> [1] 239 223 223 239 213 221 237 152 195 252 208 240 239 215
#> [15] 252
typeof(y)
#> [1] "integer"
```

Posteriormente, se realizan cinco particiones aleatorias del conjunto de líneas, con 8% este conjunto para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función *cv_random* (con los parámetros por defecto). Además, creamos los data frames vacíos *Predictions* e *Hyperparams* que servirán para guardar los valores

observados y predichos en cada ambiente y posteriormente evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
# Set seed for reproducible results
set.seed(2022)
# Unique Lines
GIDs <- unique(PhenoToy$Line)
folds <- cv_random(length(GIDs))

# A data frame that will contain the variables:
Predictions <- data.frame()
Hyperparams <- data.frame()
```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta, identificando primero las líneas correspondientes a cada conjunto;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo valores entre 0 y 1 para el hiperparámetro α y 100 como el número de lambdas generadas para ajustar los hiperparámetros (parámetro predeterminado de *lambdas_number*) con "Bayesian_optimization" como tipo de tuneo;
3. Con el modelo obtenido en (2), se predice la variable respuesta *PH* en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba.
 - b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*. Con esto, *Predictions* se formatea para que sea un argumento de la función *gs_summaries*.
5. Identificación de hiperparámetros:
 - a. *HyperparamsFold* que contiene los valores *alpha* del modelo obtenido en (2), *mse* que es el costo del modelo para cada hiperparámetro α y el número de *Fold*.
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
6. Se muestran los hiperparámetros óptimos.

```
# Model training and predictions of the ith partition
for (i in seq(folds)) {
  cat("*** Fold:", i, "***\n")
  # Identify the training and testing Line sets
  fold <- folds[[i]]
  Lines_sam_i <- GIDs[fold$training]
```

```

fold_i <- which(PhenoToy$Line %in% Lines_sam_i)

# Identify the training and testing sets
X_training <- X[fold_i, ]
X_testing <- X[-fold_i, ]
y_training <- y[fold_i]
y_testing <- y[-fold_i]

# Optional: Removing columns with no variance
var_x <- apply(X_training, 2, var)
pos_var0 <- which(var_x > 0)
X_training <- X_training[, pos_var0]
X_testing <- X_testing[, pos_var0]

# Model training
model <- generalized_linear_model(
  x = X_training,
  y = y_training,

  # Specify the hyperparameter ranges
  alpha = list(min = 0, max = 1),
  lambdas_number = 100,
  tune_type = "Bayesian_optimization",
  tune_bayes_samples_number = 5,
  tune_bayes_iterations_number = 5
)

# Testing Predictions
predictions <- predict(model, X_testing)

# Predictions for the Fold Fold
FoldPredictions <- data.frame(
  Fold = i,
  Line = PhenoToy$Line[-fold_i],
  Env = PhenoToy$Env[-fold_i],
  Observed = y_testing,
  Predicted = predictions$predicted
)
Predictions <- rbind(Predictions, FoldPredictions)

# Hyperparams for the Fold
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)

# Best hyperparams of the model
print(model$best_hyperparams)
}
#> *** Fold: 1 ***

```

```

#> *** Bayesian Optimization Tuning ***
#> Combination: 1
#> Combination: 2
#> Combination: 3
#> Combination: 4
#> Combination: 5
#> Combination: 6
#> Combination: 7
#> Combination: 8
#> Combination: 9
#> Combination: 10
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 4.3799 secs ***
#> $alpha
#> [1] 0.1276957
#>
#> $mse
#> [1] 219.0742
#>
#> $lambda
#> [1] 5.288511
#>
#> *** Fold: 2 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1
#> Combination: 2
#> Combination: 3
#> Combination: 4
#> Combination: 5
#> Combination: 6
#> Combination: 7
#> Combination: 8
#> Combination: 9
#> Combination: 10
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 4.3366 secs ***
#> $alpha
#> [1] 1
#>
#> $mse
#> [1] 230.8689
#>
#> $lambda
#> [1] 4.629971
#>
#> *** Fold: 3 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1
#> Combination: 2
#> Combination: 3

```

```

#> Combination: 4
#> Combination: 5
#> Combination: 6
#> Combination: 7
#> Combination: 8
#> Combination: 9
#> Combination: 10
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 5.2015 secs ***
#> $alpha
#> [1] 0.2633149
#>
#> $mse
#> [1] 240.4245
#>
#> $lambda
#> [1] 11.41992
#>
#> *** Fold: 4 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1
#> Combination: 2
#> Combination: 3
#> Combination: 4
#> Combination: 5
#> Combination: 6
#> Combination: 7
#> Combination: 8
#> Combination: 9
#> Combination: 10
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 4.7729 secs ***
#> $alpha
#> [1] 0.4421892
#>
#> $mse
#> [1] 249.3018
#>
#> $lambda
#> [1] 9.680093
#>
#> *** Fold: 5 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1
#> Combination: 2
#> Combination: 3
#> Combination: 4
#> Combination: 5
#> Combination: 6
#> Combination: 7

```

```

#> Combination: 8
#> Combination: 9
#> Combination: 10
#> *** Fitting Generalized Linear Model model ***
#> *** Model evaluation completed in 3.5763 secs ***
#> $alpha
#> [1] 1
#>
#> $mse
#> [1] 213.4351
#>
#> $lambda
#> [1] 3.970963

```

Predictions contienen las columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo el hiperparámetro óptimo (entre los valores posibles de α) que minimiza la función de costo *mse* con el tipo de tuneo "Bayesian_optimization", correspondiente al formato necesario para utilizar la función *gs_summaries* sobre *Predicción* en el caso de variables de conteo.

La función *gs_summaries* devuelve una lista que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```

head(Predictions)
#>   Fold      Line Env Observed Predicted
#> 1     1 CKDHL0049 EBU      252   230.9864
#> 2     1 CKDHL0049 KAK      208   216.4309
#> 3     1 CKDHL0049 KTI      240   230.9864
#> 4     1 CKDHL0108 EBU      237   230.9864
#> 5     1 CKDHL0108 KAK      219   204.4906
#> 6     1 CKDHL0108 KTI      237   231.2975

unique(Predictions$Fold)
#> [1] 1 2 3 4 5
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted Difference
#> 1 CKDHL0129 224.0000  223.1329      0.8671
#> 2 CKDHL0032 224.3333  226.0277      1.6944
#> 3 CKDHL0027 228.3333  226.0277      2.3056
#> 4 CKDHL0203 210.6667  208.2901      2.3765
#> 5 CKDHL0515 220.6667  223.0505      2.3838
#> 6 CKDHL0474 225.0000  222.3450      2.6550

# Summaries by Environment

```



```

summaries$env[, 1:8]
#>      Env      MSE MSE_SE      RMSE RMSE_SE      NRMSE NRMSE_SE
#> 1   EBU  99.4471 22.8016   9.7024   1.1522 0.8885   0.0750
#> 2   KAK 209.5109 25.1348  14.3520   0.9397 1.0443   0.0869
#> 3   KTI 293.5626 52.0005  16.7974   1.6889 1.1955   0.2495
#> 4 Global  83.4149   9.1937   9.0759   0.5107 0.9036   0.0410
#>      MAE
#> 1   8.7156
#> 2  11.9720
#> 3  14.4823
#> 4   7.5993
summaries$env[, 9:15]
#>      MAE_SE      Cor Cor_SE      Intercept Intercept_SE      Slope
#> 1  1.0676 0.1332 0.3176  1253.8447    5051.8532 -4.4227
#> 2  0.8937 0.3579 0.1676   -19.2902     104.9941  1.0466
#> 3  1.3249 0.2859 0.1008 -2949.0916    1539.6507 13.8302
#> 4  0.4587 0.4145 0.2005  -324.5755     323.2617  2.4463
#>      Slope_SE
#> 1  21.9613
#> 2   0.4966
#> 3   6.6967
#> 4   1.4429
summaries$env[, 16:19]
#>      R2  R2_SE  MAAPE  MAAPE_SE
#> 1 0.4212 0.1676 0.0373   0.0044
#> 2 0.2405 0.1289 0.0614   0.0048
#> 3 0.1224 0.0631 0.0627   0.0060
#> 4 0.3327 0.0928 0.0343   0.0022

# Summaries by Fold
summaries$fold[, 1:8]
#>      Fold      MSE MSE_SE      RMSE RMSE_SE      NRMSE NRMSE_SE
#> 1      1 241.6156 45.8108  15.4049   1.4671 0.9562   0.0433
#> 2      2 148.9419 24.8451  12.1230   0.9938 1.0904   0.1133
#> 3      3 224.2385 96.4155  13.9585   3.8340 1.2711   0.4748
#> 4      4 146.3072 51.8585  11.7273   2.0950 0.8856   0.0651
#> 5      5 243.0977 95.4675  14.8727   3.3092 1.0103   0.0498
#> 6 Global  83.4149   9.1937   9.0759   0.5107 0.9036   0.0410
#>      MAE
#> 1 13.0155
#> 2 10.9009
#> 3 11.8226
#> 4 10.5239
#> 5 12.3535
#> 6   7.5993
summaries$fold[, 9:14]
#>      MAE_SE      Cor Cor_SE      Intercept Intercept_SE      Slope
#> 1 0.9405 0.1825 0.2330   492.2746     889.8283  -1.1550
#> 2 0.8886 -0.0530 0.1279  4221.4380    4871.5812 -17.3777
#> 3 3.3668 0.3579 0.2426   67.5271     108.6203   0.6935

```

```
#> 4 2.0757 0.6384 0.2530 -8362.5864 4689.4888 37.4006
#> 5 2.6138 0.1692 0.3996 723.7849 3282.9842 -2.1378
#> 6 0.4587 0.4145 0.2005 -324.5755 323.2617 2.4463
summaries$fold[, 15:19]
#> Slope_SE R2 R2_SE MAAPE MAAPE_SE
#> 1 3.8406 0.1418 0.0609 0.0613 0.0065
#> 2 21.1366 0.0355 0.0083 0.0494 0.0071
#> 3 0.4706 0.2458 0.2261 0.0538 0.0150
#> 4 20.3773 0.5355 0.2647 0.0472 0.0080
#> 5 14.2842 0.3480 0.0314 0.0572 0.0123
#> 6 1.4429 0.3327 0.0928 0.0343 0.0022
```

Además, Hperparams contiene las columnas *alpha*, *mse* y *Fold*, donde el valor de *mse* corresponde a caa combinación de α los valores de la partición y, ordendos de menor a mayor dentro de cada partición.

```
# First rows of Hyperparams
head(Hyperparams)
#>      alpha      mse Fold
#> 2 0.12769568 219.0742    1
#> 9 0.15952848 219.9465    1
#> 6 0.22099039 221.3266    1
#> 5 0.05200501 222.9618    1
#> 7 0.36046755 223.5197    1
#> 3 0.50889883 225.1531    1

# Last rows of Hyperparams
tail(Hyperparams)
#>      alpha      mse Fold
#> 104 1.0000000 213.4351    5
#> 14 0.9548791 213.5300    5
#> 34 0.7194566 214.2109    5
#> 44 0.6642231 214.4370    5
#> 24 0.5168080 215.2334    5
#> 54 0.3762821 216.5210    5
```

3.5 Ejemplo multivariados con respuestas continuas con optimización bayesiana con validación cruzada 7-fold con *Env* + *G* el predictor

Este ejemplo evalúa un modelo lineal generalizado con validación cruzada 7-fold, para dos respuestas continuas, usando el efecto ambiente y la matriz *G* como predictores y usando "Bayesian_optimization" como tipo de tuneo para el hiperparámetro α cuyas opciones están entre 0 y uno.

En este ejemplo, el conjunto de datos utilizado es *MaizeToy* y el objetivo es predecir las variables continuas *Yield* y *ASI* del marco de datos *PhenoToy* utilizando la matriz de diseño de la variable *PhenoToy Env* y la matriz como *G* predictores; por lo que identificamos las variables predictoras y de respuesta como *X* y *y* respectivamente.

```

# Load the data
load("MaizeToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env & G
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %*% Geno

# Predictor and Response Variables
X <- cbind(Env, LineG)
y <- PhenoToy[, c("Yield", "ASI")]

```

Posteriormente hacemos 7 particiones aleatorias, con la ayuda de la función *cv_kfold*. Además creamos los data frames vacíos *PredictionsASI*, *PredictionsYield* e *Hyperparams* que servirán para guardar los valores observados y predichos en cada ambiente y posteriormente evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```

# Set seed for reproducible results
set.seed(2022)

folds <- cv_kfold(records_number = nrow(X), k = 7)

# Data frames that will contain the variables:
PredictionsYield <- data.frame()
PredictionsASI <- data.frame()
Hyperparams <- data.frame()

```

Posteriormente, se seguirá el siguiente proceso **para cada partición y para cada variable de respuesta** :

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo valores entre 0 y 1 para el hiperparámetro α y 100 como el número de lambdas generadas para ajustar los hiperparámetros (parámetro predeterminado de *lambdas_number*) con "Bayesian_optimization" como tipo de tuneo;
3. Con el modelo obtenido en (2), se predice la variable respuesta en el conjunto de prueba, con el fin de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:

- a. Se crean los data frames *FoldPredictionsASI* y *FoldPredictionYield* que contienen las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba y para cada variable de respuesta respectiva.
 - b. Cada fila de *FoldPredictionASI* se agrega al data frame *PredictionsASI* ; y cada fila de *FoldPredictionYield* se agrega al data frame de *PredictionsYield*.
5. Identificación de hiperparámetros:
 - a. *HyperparamsFold* que contiene los valores *alpha* del modelo obtenido en (2), *multivariate_loss* , que es el costo del modelo para cada hiperparámetro α y el número de *pliege* .
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
6. Se muestran los hiperparámetros óptimos.

```
# Model training and predictions of the ith partition
```

```
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")
  fold <- folds[[i]]

  # Identify the training and testing sets
  X_training <- X[fold$training, ]
  X_testing <- X[fold$testing, ]
  y_training <- y[fold$training, ]
  y_testing <- y[fold$testing, ]

  # Model training
  model <- generalized_linear_model(
    x = X_training,
    y = y_training,

    # Specify the hyperparameter ranges
    alpha = list(min = 0, max = 1),
    lambdas_number = 100,
    tune_type = "Bayesian_optimization",
    tune_bayes_samples_number = 5,
    tune_bayes_iterations_number = 5
  )

  # Testing Predictions
  predictions <- predict(model, X_testing)

  # Predictions of Yield for the Fold
  FoldPredictionsYield <- data.frame(
    Fold = i,
    Line = PhenoToy$Line[fold$testing],
    Env = PhenoToy$Env[fold$testing],
    Observed = y_testing$Yield,
    Predicted = predictions$Yield$predicted
  )
}
```

```

PredictionsYield <- rbind(PredictionsYield, FoldPredictionsYield)

# Predictions of ASI for the Fold
FoldPredictionsASI <- data.frame(
  Fold = i,
  Line = PhenoToy$Line[fold$testing],
  Env = PhenoToy$Env[fold$testing],
  Observed = y_testing$ASI,
  Predicted = predictions$ASI$predicted
)
PredictionsASI <- rbind(PredictionsASI, FoldPredictionsASI)

# Hyperparams for the Fold
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)

# Best hyperparams of the model
print(model$best_hyperparams)
}
#> *** Fold: 1 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1
#> Combination: 2
#> Combination: 3
#> Combination: 4
#> Combination: 5
#> Combination: 6
#> Combination: 7
#> Combination: 8
#> Combination: 9
#> Combination: 10
#> *** Fitting Multivariate Generalized Linear Model model ***
#> *** Model evaluation completed in 4.9565 secs ***
#> $alpha
#> [1] 0.4892824
#>
#> $multivariate_loss
#> [1] 1.428865
#>
#> $lambda
#> [1] 0.1545847
#>
#> *** Fold: 2 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1
#> Combination: 2
#> Combination: 3
#> Combination: 4
#> Combination: 5

```

```

#> Combination: 6
#> Combination: 7
#> Combination: 8
#> Combination: 9
#> Combination: 10
#> *** Fitting Multivariate Generalized Linear Model model ***
#> *** Model evaluation completed in 4.7059 secs ***
#> $alpha
#> [1] 1
#>
#> $multivariate_loss
#> [1] 1.410177
#>
#> $lambda
#> [1] 0.1243676
#>
#> *** Fold: 3 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1
#> Combination: 2
#> Combination: 3
#> Combination: 4
#> Combination: 5
#> Combination: 6
#> Combination: 7
#> Combination: 8
#> Combination: 9
#> Combination: 10
#> *** Fitting Multivariate Generalized Linear Model model ***
#> *** Model evaluation completed in 4.7388 secs ***
#> $alpha
#> [1] 0.3740331
#>
#> $multivariate_loss
#> [1] 1.472595
#>
#> $lambda
#> [1] 0.1562236
#>
#> *** Fold: 4 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1
#> Combination: 2
#> Combination: 3
#> Combination: 4
#> Combination: 5
#> Combination: 6
#> Combination: 7
#> Combination: 8
#> Combination: 9

```

```

#> Combination: 10
#> *** Fitting Multivariate Generalized Linear Model model ***
#> *** Model evaluation completed in 4.7898 secs ***
#> $alpha
#> [1] 1
#>
#> $multivariate_loss
#> [1] 1.732025
#>
#> $lambda
#> [1] 0.1329653
#>
#> *** Fold: 5 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1
#> Combination: 2
#> Combination: 3
#> Combination: 4
#> Combination: 5
#> Combination: 6
#> Combination: 7
#> Combination: 8
#> Combination: 9
#> Combination: 10
#> *** Fitting Multivariate Generalized Linear Model model ***
#> *** Model evaluation completed in 5.0905 secs ***
#> $alpha
#> [1] 0.9232725
#>
#> $multivariate_loss
#> [1] 1.713672
#>
#> $lambda
#> [1] 0.7181523
#>
#> *** Fold: 6 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1
#> Combination: 2
#> Combination: 3
#> Combination: 4
#> Combination: 5
#> Combination: 6
#> Combination: 7
#> Combination: 8
#> Combination: 9
#> Combination: 10
#> *** Fitting Multivariate Generalized Linear Model model ***
#> *** Model evaluation completed in 4.2197 secs ***
#> $alpha

```

```

#> [1] 0.4392586
#>
#> $multivariate_loss
#> [1] 1.599469
#>
#> $lambda
#> [1] 0.1426964
#>
#> *** Fold: 7 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1
#> Combination: 2
#> Combination: 3
#> Combination: 4
#> Combination: 5
#> Combination: 6
#> Combination: 7
#> Combination: 8
#> Combination: 9
#> Combination: 10
#> *** Fitting Multivariate Generalized Linear Model model ***
#> *** Model evaluation completed in 6.0114 secs ***
#> $alpha
#> [1] 0.5477275
#>
#> $multivariate_loss
#> [1] 1.490304
#>
#> $lambda
#> [1] 0.09944179

```

Repitiendo este proceso para cada partición, los *data frames* *PredictionsASI* y *PredictionsYield* contienen las columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición en su respectiva variable de respuesta, donde las predicciones se realizan eligiendo el óptimo hiperparámetro (entre los posibles valores de α) que minimiza la función de costo con el tipo de tuneo "Bayesian Optimization", correspondiente al formato necesario para usar la función *gs_summaries* en estas predicciones en el caso de variables continuas.

La función *gs_summaries* devuelve una lista que contiene tres *data frames* correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```

head(PredictionsASI)
#>   Fold      Line Env Observed Predicted
#> 1    1 CKDHL0032 KTI      1.3  2.246198
#> 2    1 CKDHL0049 EBU      2.3  1.697267
#> 3    1 CKDHL0050 EBU      3.1  1.614085
#> 4    1 CKDHL0052 KTI      2.0  2.216580
#> 5    1 CKDHL0085 KTI      2.7  2.280677
#> 6    1 CKDHL0097 KTI      2.0  2.261327

```



```

unique(PredictionsASI$Fold)
#> [1] 1 2 3 4 5 6 7
head(PredictionsYield)
#>   Fold      Line Env Observed Predicted
#> 1     1 CKDHL0032 KTI      6.24  6.085749
#> 2     1 CKDHL0049 EBU      4.72  6.426266
#> 3     1 CKDHL0050 EBU      4.98  6.716740
#> 4     1 CKDHL0052 KTI      7.20  6.123782
#> 5     1 CKDHL0085 KTI      7.41  6.079875
#> 6     1 CKDHL0097 KTI      4.45  6.075007
unique(PredictionsYield$Fold)
#> [1] 1 2 3 4 5 6 7
# Summaries
summariesASI <- gs_summaries(PredictionsASI)
summariesYield <- gs_summaries(PredictionsYield)

# Elements of summaries
names(summariesASI)
#> [1] "line" "env" "fold"

# Summaries by Line
head(summariesASI$line)
#>      Line Observed Predicted Difference
#> 1 CKDHL0491  1.9000    1.9063    0.0063
#> 2 CKDHL0150  1.8667    1.8752    0.0086
#> 3 CKDHL0136  1.7333    1.7680    0.0347
#> 4 CKDHL0052  1.8000    1.7374    0.0626
#> 5 CKDHL0502  1.6667    1.7342    0.0675
#> 6 CKDHL0050  1.9333    1.8591    0.0742
head(summariesYield$line)
#>      Line Observed Predicted Difference
#> 1 CKDHL0515  6.0100    6.0038    0.0062
#> 2 CKDHL0050  5.9300    5.9219    0.0081
#> 3 CKDHL0136  5.8800    5.8718    0.0082
#> 4 CKDHL0054  5.9233    5.9099    0.0134
#> 5 CKDHL0027  5.8800    5.8357    0.0443
#> 6 CKDHL0160  5.7633    5.8561    0.0927

# Summaries by Environment
summariesASI$env[, 1:9]
#>      Env      MSE MSE_SE  RMSE RMSE_SE  NRMSE NRMSE_SE  MAE
#> 1    EBU 0.8126 0.2640 0.8476 0.1253 1.6831 0.3617 0.7414
#> 2    KAK 0.8931 0.2434 0.8853 0.1351 0.9874 0.0467 0.6744
#> 3    KTI 0.7200 0.2153 0.8049 0.1097 1.5099 0.2791 0.6875
#> 4 Global 0.8141 0.1519 0.8819 0.0777 1.0535 0.0620 0.7025
#>      MAE_SE
#> 1 0.1079
#> 2 0.0901
#> 3 0.1123
#> 4 0.0633

```

```

summariesYield$env[, 1:9]
#>      Env      MSE MSE_SE      RMSE RMSE_SE      NRMSE NRMSE_SE      MAE
#> 1   EBU 0.7563 0.1954 0.8264 0.1107 2.0563 0.5524 0.7191
#> 2   KAK 0.5880 0.1689 0.6966 0.1308 1.1805 0.1862 0.6240
#> 3   KTI 1.2137 0.2697 1.0389 0.1497 1.3632 0.4462 0.9282
#> 4 Global 0.8185 0.1711 0.8675 0.1049 0.9180 0.0521 0.7301
#>      MAE_SE
#> 1 0.1057
#> 2 0.1225
#> 3 0.1244
#> 4 0.0986

# Summaries by Fold
summariesASI$fold[, 1:8]
#>      Fold      MSE MSE_SE      RMSE RMSE_SE      NRMSE NRMSE_SE      MAE
#> 1      1 0.3299 0.1201 0.5554 0.1035 1.2763 0.2075 0.4770
#> 2      2 2.0735 0.1161 1.4389 0.0398 1.6078 0.3767 1.2155
#> 3      3 0.4848 0.1525 0.6792 0.1085 1.1952 0.2448 0.6142
#> 4      4 0.7763 0.1785 0.8674 0.1093 1.8020 0.8785 0.6988
#> 5      5 0.6070 0.1340 0.7680 0.0925 1.0093 0.0527 0.5842
#> 6      6 0.5635 0.1315 0.7386 0.0949 0.9680 0.0597 0.6222
#> 7      7 0.8249 0.3290 0.8738 0.1753 1.8295 0.4803 0.6959
#> 8 Global 0.8141 0.1519 0.8819 0.0777 1.0535 0.0620 0.7025
summariesYield$fold[, 1:8]
#>      Fold      MSE MSE_SE      RMSE RMSE_SE      NRMSE NRMSE_SE      MAE
#> 1      1 1.3056 0.1944 1.1358 0.0884 1.0599 0.0732 1.0239
#> 2      2 0.6054 0.3407 0.7206 0.2076 1.9297 1.1350 0.6023
#> 3      3 1.2304 0.3085 1.0925 0.1358 1.9396 1.1162 0.9852
#> 4      4 0.1614 0.0976 0.3559 0.1318 0.9867 0.1366 0.3109
#> 5      5 1.2714 0.4816 1.0805 0.2279 1.2232 0.3520 0.9583
#> 6      6 0.6237 0.3400 0.7340 0.2060 1.3162 0.1371 0.6265
#> 7      7 0.7710 0.2323 0.8585 0.1305 2.2378 0.8979 0.7925
#> 8 Global 0.8185 0.1711 0.8675 0.1049 0.9180 0.0521 0.7301

```

Además, Hyperparams contiene las columnas *alpha*, *multivariate_loss* *Fold*, donde el valor de *multivariate_loss* corresponde al costo del modelo para cda combinación de los valores de α y partición, ordenados de menor a mayor dentro de cada partición.

```

# First rows of Hyperparams
head(Hyperparams)
#>      alpha multivariate_loss Fold
#> 10 0.4892824          1.428865    1
#> 7  0.4986157          1.428910    1
#> 9  0.5307280          1.429330    1
#> 2  0.6214780          1.431456    1
#> 1  0.3743163          1.433084    1
#> 3  0.7495342          1.434167    1

# Last rows of Hyperparams
tail(Hyperparams)

```

```
#>      alpha multivariate_loss Fold
#> 76  0.75810720      1.491537    7
#> 96  0.87063994      1.492606    7
#> 106 0.98082581      1.493417    7
#> 56  0.09444095      1.522727    7
#> 16  0.08940186      1.523738    7
#> 46  0.04669136      1.541265    7
```

3.6 Ejemplo con kernels con Grid search y particiones aleatorias

Este ejemplo evalúa un modelo lineal generalizado con cinco particiones aleatorias, con 20% los datos para el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de cada partición (los parámetros predeterminados de la función `cv_random`), para una respuesta continua, utilizando la matriz de diseño de *PhenoToy* Env variable, la matriz descrita *G* anteriormente y la matriz de diseño de la interacción entre estas dos, como predictores; además de utilizar "Grid Search" como 1 tipo de tuneo del hiperparámetro α cuyas opciones son 0, 0.25, 0.50 y 0.75. Todo esto para los tipos de Kernel: "Linear", "Polinomial", "Sigmoid", "Gaussian", "Exponential", "Arc_cosine" y "Arc_cosine_L".

En este ejemplo, el conjunto de datos utilizado es *ChickpeaToy* y el objetivo es predecir la variable continua *Biomass* del data frame *PhenoToy** utilizando la matriz de diseño de la variable *PhenoToy* Env, la matriz descrita *G* anteriormente y la matriz de diseño de la interacción entre estos dos, como predictores; por lo que identificamos las variables predictoras y de respuesta como *X* y *y* respectivamente.

```
# Load the data
load("ChickpeaToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env, G & GE
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LinexGeno <- Line %**% Geno
LinexGenoxEnv <- model.matrix(~ 0 + LinexGeno:Env)

# Predictor and Response Variables
X <- cbind(Env, LinexGeno, LinexGenoxEnv)
y <- PhenoToy$Biomass

dim(X)
#> [1] 180 216
print(y[1:7])
#> [1] 235.0000 392.5000 249.3333 358.6667 309.6667 198.0000
#> [7] 171.0000
```

```
typeof(y)
#> [1] "double"
```

Tenga en cuenta que la variable de respuesta *y* es una variable continua (un vector con elementos de tipo "double"), lo cual es importante para que el modelo se entrene automáticamente para una variable continua.

A diferencia de los ejemplos anteriores, ahora buscamos evaluar el modelo para cada tipo de kernel mencionado anteriormente. Por ello, creamos un vector en el que indicamos los tipos de kernel que queremos aplicar a la matriz *X*. Además, creamos las listas vacías *PredictionsAll*, *TimesAll*, *HyperparamsAll* y *SummariesAll* que se utilizarán para guardar las predicciones, los tiempos de ejecución, los hiperparámetros y los resúmenes de cada modelo entrenado, es decir, para cada tipo de kernel; lo que a su vez servirá para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
kernels <- c(
  "linear",
  "polynomial",
  "sigmoid",
  "Gaussian",
  "exponential",
  "arc_cosine",
  "Arc_cosine_L"
)

# Example: Apply the Linear Kernel to the data
X_Linear <- kernelize(X, kernel = kernels[1])

# Note that X_Linear is a square matrix
dim(X_Linear)
#> [1] 180 180

# Empty lists that will contain Predictions,
# Times of execution & Summaries for each type of kernel
PredictionsAll <- list()
TimesAll <- list()
HyperparamsAll <- list()
SummariesAll <- list()
```

Subsequently, the following process will be followed **for each type of Kernel**:

Posteriormente, se seguirá el siguiente proceso **para cada tipo de Kernel**:

1. identifique la variable *arc_deep* con el valor 2. Si el tipo de Kernel es "Arc_cosine_L", el valor de la variable *arc_deep* se cambia a 3 y el *kernel_type* se identifica como "Arc_cosine"; de lo contrario, *kernel_type* se identifica como el kernel predeterminado.

2. El tipo de kernel establecido en (1) se aplica a la matriz de datos X , asignando al argumento *arc_cosine_deep* el valor establecido en la variable *arc_deep*. Tenga en cuenta que el argumento *arc_cosine_deep* se ignora si el tipo de kernel no es *Arc_cosine*.
3. Luego realizamos cinco particiones aleatorias, con 80% los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función *cv_random*.
4. *Predictions*, *Times* e *Hyperparams* que se utilizarán para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función *gs_summaries*, además de guardar los tiempos de ejecución de cada modelo entrenado para cada partición.
5. **Para cada partición:**
 1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
 2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo los valores 0, 0.25, 0.50, 0.75 y 1 para el hiperparámetro α y 100 como el número de lambdas generadas para tunear los hiperparámetros (parámetro por defecto de *lambdas_number*) con "Grid Search" como tipo de sintonización (parámetro por defecto de *tune_type*);
 3. Con el modelo obtenido en (2), se predice la variable respuesta y en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
 4. Identificación de las predicciones del conjunto de prueba: Se crea el data frame *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba. Además, cada fila de *FoldPrediction* se agrega al data frame *Predictions*.
 5. Identificación del tiempo de ejecución del entrenamiento del modelo obtenido en (4): Se crea el data frame *FoldTime* que contiene la columna que especifica el tipo de *Kernel* utilizado para entrenar el modelo, el número de *Fold* y la columna *Minutos que indica el tiempo de ejecución*, en minutos, del entrenamiento del modelo obtenido en (2). Además, cada fila de *FoldTime* se agrega al data frame *Times*.
 6. Identificación de hiperparámetros; Se crea el data frame *HyperparamsFold* que contiene los valores *alpha* del modelo obtenido en (2), *loss*, que es el costo del modelo para cada hiperparámetro α y el número de *Fold*. Además, cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.

Predictions contiene las columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo el hiperparámetro óptimo (entre los valores posibles de α) que minimiza la

función de costo con el tipo de tuneo "Grid Search", correspondiente al formato necesario para utilizar la función *gs_summaries* sobre estas predicciones en el caso de variables continuas.

6. Predictions y con la ayuda de la función *gs_summaries*, evaluamos la capacidad predictiva del modelo entrenado para el tipo de kernel especificado. La función *gs_summaries* devuelve una lista que identificamos como *summaries* y que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.
7. Finalmente, se crea un elemento con el nombre de kernel especificado en cada una de las listas *PredictionsAll*, *TimesAll*, *HyperparamsAll* y *SummariesAll*, que corresponden a los marcos de datos de las listas *Predictions*, *Times*, *Hyperparams* y *summaries*, respectivamente.

```
for (kernel in kernels) {
  cat("*** Kernel:", kernel, "***\n")

  # Identify the arc_deep and the kernel
  arc_deep <- 2
  if (kernel == "Arc_cosine_L") {
    arc_deep <- 3
    kernel <- "arc_cosine"
  } else {
    kernel <- kernel
  }

  # Compute the kernel
  X <- kernelize(X, kernel = kernel, arc_cosine_deep = arc_deep)

  # Random Partition
  set.seed(2022)
  folds <- cv_random(
    records_number = nrow(X),
    folds_number = 5,
    testing_proportion = 0.2
  )

  # Empty data frames that will contain Predictions, Times
  # of execution & Summaries for each partition
  Predictions <- data.frame()
  Times <- data.frame()
  Hyperparams <- data.frame()

  for (i in seq_along(folds)) {
    cat("\t*** Fold:", i, "***\n")
    fold <- folds[[i]]

    # Identify the training and testing sets
    X_training <- X[fold$training, ]
```

```

X_testing <- X[fold$testing, ]
y_training <- y[fold$training]
y_testing <- y[fold$testing]

# Model training
model <- generalized_linear_model(
  x = X_training,
  y = y_training,

  # Specify the hyperparameters values
  alpha = c(0, 0.25, 0.50, 0.75, 1),
  lambdas_number = 100,
  tune_folds_number = 5,
  tune_type = "grid_search",
  tune_grid_proportion = 0.8,

  # In this example the iterations won't be shown
  verbose = FALSE
)

# Testing Predictions
predictions <- predict(model, X_testing)

# Predictions for the Fold Fold
FoldPredictions <- data.frame(
  Fold = i,
  Line = PhenoToy$Line[fold$testing],
  Env = PhenoToy$Env[fold$testing],
  Observed = y_testing,
  Predicted = predictions$predicted
)
Predictions <- rbind(Predictions, FoldPredictions)

# Execution times
FoldTime <- data.frame(
  kernel = kernel,
  Fold = i,
  Minutes = as.numeric(model$execution_time, units = "mins")
)
Times <- rbind(Times, FoldTime)

# Hyperparams for the Fold
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)
}

# Summaries of the Folds
summaries <- gs_summaries(Predictions)

```

```

# Predictions, Times of execution & Summaries for the
# specified Kernel
PredictionsAll[[kernel]] <- Predictions
TimesAll[[kernel]] <- Times
HyperparamsAll[[kernel]] <- Hyperparams
SummariesAll[[kernel]] <- summaries
}
#> *** Kernel: linear ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: polynomial ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: sigmoid ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: Gaussian ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: exponential ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: arc_cosine ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: Arc_cosine_L ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***

```


Recordando que este proceso se realizó para cada tipo de kernel, cada una de las listas *PredictionsAll*, *TimesAll*, *HyperparamsAll* y *SummariesAll* contiene las predicciones, los tiempos de ejecución, las combinaciones de los hiperparámetros (en este caso *alpha*) y los resúmenes, respectivamente, para cada tipo de kernel aplicado a la matriz de datos *X*. A modo de ejemplo, los resultados obtenidos para el tipo de kernel “Linear” se muestran a continuación:

```
# Predictions for the Linear Kernel
head(PredictionsAll$linear)
#>   Fold      Line Env Observed Predicted
#> 1     1 ICCV97301   6 431.0000  329.4303
#> 2     1 ICCV04103   1 370.3333  243.2823
#> 3     1 ICCV05109   4 241.3333  243.2576
#> 4     1 ICCV00402   7 198.0000  274.4901
#> 5     1 ICCV09114   4 271.6667  235.4623
#> 6     1 ICCV03102   2 612.5000  328.8074

# Times of execution for the Linear Kernel
TimesAll$linear
#>   kernel Fold      Minutes
#> 1 linear     1 0.011231164
#> 2 linear     2 0.007415930
#> 3 linear     3 0.007502739
#> 4 linear     4 0.006925158
#> 5 linear     5 0.007082478

# Elements of SummariesAll
names(SummariesAll)
#> [1] "linear"      "polynomial"  "sigmoid"     "Gaussian"
#> [5] "exponential" "arc_cosine"

# Elements of summaries for the Linear Kernel
names(SummariesAll$linear)
#> [1] "line" "env"  "fold"

# Summaries by Line
head(SummariesAll$linear$line)
#>   Line Observed Predicted Difference
#> 1 ICCV10316 247.4000  246.6886    0.7114
#> 2 ICCV96331 297.8667  292.9877    4.8790
#> 3 ICCV03309 251.8667  258.1340   -6.2673
#> 4 ICCV03407 272.9000  265.3837    7.5163
#> 5 ICCV10112 293.8750  286.2745    7.6005
#> 6 ICCV01301 256.4444  248.2266    8.2178

# Summaries by Environment
SummariesAll$linear$env[, 1:8]
#>   Env      MSE  MSE_SE  RMSE RMSE_SE  NRMSE NRMSE_SE
#> 1     1 6875.430 1311.8864 81.1641  8.4827 1.0508  0.0454
```

```

#> 2      2 7519.932 2301.3137 83.0753 12.4342 0.8851 0.0416
#> 3      4 1721.267 333.3951 40.5592 4.3652 1.0662 0.1278
#> 4      5 2233.537 633.7437 45.1552 6.9740 0.9060 0.0853
#> 5      6 3602.056 989.7324 57.3608 8.8289 1.7155 0.4246
#> 6      7 2203.865 795.9604 44.1320 8.0036 1.6168 0.3223
#> 7 Global 3935.853 868.7861 61.0246 7.2776 0.7308 0.0419
#>      MAE
#> 1 68.2023
#> 2 63.9285
#> 3 35.6121
#> 4 41.4906
#> 5 48.1576
#> 6 36.2816
#> 7 46.3177
SummariesAll$linear$env[, 9:15]
#>      MAE_SE      Cor Cor_SE Intercept Intercept_SE Slope Slope_SE
#> 1 7.8350 0.2389 0.0573 -160.0667 190.9703 1.6771 0.8355
#> 2 5.7602 0.3525 0.2590 -228.6208 1100.7638 1.6550 3.2082
#> 3 3.8357 0.4209 0.1431 102.2344 48.5041 0.5991 0.1901
#> 4 7.2815 0.5780 0.0664 -196.5261 216.1002 2.1673 0.7673
#> 5 8.1528 0.0205 0.2690 99.2030 271.6476 0.7415 0.7963
#> 6 7.1674 0.5851 0.1641 48.5085 40.1714 0.6405 0.2230
#> 7 4.9082 0.6950 0.0366 -69.8135 32.4841 1.2536 0.1327
SummariesAll$linear$env[, 16:19]
#>      R2 R2_SE MAAPE MAAPE_SE
#> 1 0.0702 0.0265 0.2781 0.0381
#> 2 0.3925 0.1115 0.1797 0.0135
#> 3 0.2590 0.1204 0.1443 0.0141
#> 4 0.3506 0.0711 0.1697 0.0360
#> 5 0.2900 0.1557 0.1368 0.0218
#> 6 0.4501 0.1708 0.2015 0.0402
#> 7 0.4884 0.0523 0.1733 0.0197

# Summaries by Fold
SummariesAll$linear$fold[, 1:8]
#>      Fold      MSE      MSE_SE      RMSE RMSE_SE      NRMSE NRMSE_SE
#> 1      1 6765.752 2319.9780 75.8536 14.2266 1.3532 0.2738
#> 2      2 3508.521 821.8620 57.0274 7.1609 1.4840 0.3410
#> 3      3 2173.220 766.0682 43.0508 7.9981 0.9225 0.0510
#> 4      4 4786.290 1035.8978 67.0221 7.6724 1.3370 0.3009
#> 5      5 2896.291 1075.7867 49.9181 8.9942 1.0332 0.1027
#> 6 Global 3935.853 868.7861 61.0246 7.2776 0.7308 0.0419
#>      MAE
#> 1 62.1932
#> 2 46.1275
#> 3 35.5210
#> 4 58.1626
#> 5 42.7231
#> 6 46.3177
SummariesAll$linear$fold[, 9:15]

```

```

#>      MAE_SE      Cor Cor_SE Intercept Intercept_SE      Slope
#> 1 10.2800 0.0119 0.1393  637.8885      648.2703 -0.8256
#> 2  5.2806 0.1724 0.2578  196.9770      74.6426  0.3573
#> 3  6.1457 0.4110 0.1122 -645.4735     459.0311  3.2158
#> 4  5.8277 0.4546 0.1310 -152.8869     246.3478  1.2742
#> 5  6.9960 0.7124 0.0804 -315.8948     209.6243  1.9104
#> 6  4.9082 0.6950 0.0366  -69.8135      32.4841  1.2536
#>      Slope_SE
#> 1    1.9281
#> 2    0.2047
#> 3    1.3669
#> 4    0.7664
#> 5    0.6210
#> 6    0.1327
SummariesAll$linear$fold[, 16:19]
#>      R2      R2_SE  MAAPE  MAAPE_SE
#> 1 0.0972 0.0559 0.2129  0.0287
#> 2 0.3488 0.1359 0.1809  0.0323
#> 3 0.2319 0.0860 0.1242  0.0147
#> 4 0.2924 0.1265 0.2395  0.0342
#> 5 0.5398 0.0941 0.1675  0.0296
#> 6 0.4884 0.0523 0.1733  0.0197

```

Además, los elementos de la lista *HyperparamsAll* contienen las columnas *alpha*, *loss* y *Fold*, donde el valor de la columna de pérdida corresponde al costo del model para cada combinación de los α valores de partición y, ordenados de menor a mayor dentro de cada partición.

```

# First rows of Hyperparams
head(HyperparamsAll$linear)
#>      alpha      Loss Fold
#> 2    0.25 2965.373    1
#> 4    0.75 2999.160    1
#> 5    1.00 3001.254    1
#> 1    0.00 3060.944    1
#> 21    0.25 3994.289    2
#> 3    0.50 4033.894    2

# Last rows of Hyperparams
tail(HyperparamsAll$linear)
#>      alpha      Loss Fold
#> 42    0.75 4153.730    4
#> 53    1.00 4154.292    4
#> 54    1.00 3846.198    5
#> 33    0.50 3849.834    5
#> 22    0.25 3885.358    5
#> 14    0.00 3897.018    5

```

3.7 Ejemplo de con Sparse Kernels con Grid search y particiones aleatorias

Este ejemplo evalúa un modelo lineal generalizado con cinco particiones aleatorias, con 20% los datos para el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de cada partición (los parámetros predeterminados de la función `cv_random`), para una respuesta continua, utilizando la matriz de diseño de *PhenoToy* Env variable, la matriz descrita *G* anteriormente y la matriz de diseño de la interacción entre estas dos, como predictores; además de utilizar "Grid Search" como 1 tipo de tuneo del hiperparámetro α cuyas opciones son 0, 0.25, 0.50 y 0.75. Todo esto con los llamados "Sparse Kernel Methods", con las posibles combinaciones entre los tipos de Kernel "Sparse_Gaussian" y "Sparse_Arc_cosine" con las proporciones 0.5,0.6,0.7,0.8,0.9 y 1.

En este ejemplo, el conjunto de datos utilizado es *GroundnutToy* y el objetivo es predecir la variable continua *PYPP* del marco de datos *PhenoToy** utilizando la matriz de diseño de la variable *PhenoToy* Env, la matriz descrita *G* anteriormente y la matriz de diseño de la interacción entre estos dos, como predictores; por lo que identificamos las variables predictoras y de respuesta como *X* y *y* respectivamente.

```
# Load the data
load("GroundnutToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy
# Data preparation of Env, G & GE
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LinexGeno <- Line %*% Geno
LinexGenoEnv <- model.matrix(~ 0 + LinexGeno:Env)

# Predictor and Response Variables
X <- cbind(Env, LinexGeno, LinexGenoEnv)
y <- PhenoToy$PYPP

dim(X)
#> [1] 120 154
print(y[1:7])
#> [1] 13.45  6.04  7.18  7.60 12.29  3.71  9.24
typeof(y)
#> [1] "double"
```

Tenga en cuenta que la variable de respuesta *y* es una variable continua (un vector con elementos de tipo "doble"), lo cual es importante para que el modelo se entrene automáticamente para una variable continua.

A diferencia de los ejemplos anteriores, ahora buscamos evaluar el modelo para cada una de las posibles combinaciones entre los tipos de Kernel "Sparse_Gaussian" y

“Sparse_Arc_cosine” con las proporciones 0.5,0.6,0.7,0.8,0.9 y 1. Por eso creamos un vector llamado *kernels* en el que indicamos los tipos de kernels que queremos aplicar a los de la matriz *X* y otro vector llamado *lines_proportions*. Además, creamos las listas vacías *PredictionsAll*, *TimesAll*, *HyperparamsAll* y *SummariesAll* que se utilizarán para guardar las predicciones, los tiempos de ejecución, los hiperparámetros y los resúmenes de cada modelo entrenado, es decir, para cada combinación entre tipo de kernel y proporción de *líneas* utilizadas, que a su vez servirán para guardar los valores observados y predichos en cada ambiente y para posteriormente evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
kernels <- c("Sparse_Gaussian", "Sparse_Arc_cosine")
lines_proportions <- c(0.5, 0.6, 0.7, 0.8, 0.9, 1)

# Example: Apply the "Sparse_Gaussian" Kernel to the data
X_Linear <- kernelize(
  X,
  kernel = kernels[1],
  rows_proportion = lines_proportions[1]
)
# Note that X_Linear is an square matrix
dim(X_Linear)
#> [1] 120 60

# Empty lists that will contain Predictions,
# Times of execution & Summaries for each type of kernel
PredictionsAll <- list()
TimesAll <- list()
HyperparamsAll <- list()
SummariesAll <- list()
```

Posteriormente, se siguió el siguiente proceso **para cada tipo de Kernel y para cada proporción de líneas**:

1. El conjunto de tipos de kernel se aplica a la matriz de datos *X*, asignando el valor numérico a la2 El argumento *arc_cosine_deep* y la proporción de líneas establecen el valor en el argumento de proporción de *filas*.
2. Luego realizamos cinco particiones aleatorias, con 80% los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función *cv_random*.
3. *Predictions*, *Times* e *Hyperparams* que se utilizarán para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función *gs_summaries*, además de guardar los tiempos de ejecución de cada modelo entrenado para cada partición.
4. **Para cada partición:**

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
 2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo los valores 0, 0.25, 0.50, 0.75 y 1 para el hiperparámetro α y 100 como el número de lambdas generadas para ajustar los hiperparámetros (parámetro predeterminado de *lambdas_number*) con "Optimización bayesiana" como tipo de tuneo;
 3. Con el modelo obtenido en (3), se predice la variable respuesta y en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
 4. Identificación de las predicciones del conjunto de prueba: Se crea el data frame *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba. Además, cada fila de *FoldPrediction* se agrega al data frame *Predictions*.
 5. Identificación del tiempo de ejecución del entrenamiento del modelo obtenido en (2): Se crea el data frame *FoldTime* que contiene la columna que especifica el tipo de *Kernel* utilizado para entrenar el modelo, el número de *Fold* y la columna *Minutos* que indica el tiempo de ejecución, en minutos, del entrenamiento del modelo obtenido en (2). Además, cada fila de *FoldTime* se agrega al data frame *Times*.
 6. Identificación de hiperparámetros; Se crea el data frame *HyperparamsFold* que contiene los valores *alpha* del modelo obtenido en (2), *loss*, que es el costo del modelo para cada hiperparámetro α y el número de *Fold*. Además, cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
- *Predictions* contiene las columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo el hiperparámetro óptimo (entre los valores posibles de α) que minimiza la función de costo con el Tipo de tuneo "Bayesian Optimization", correspondiente al formato necesario para utilizar la función *gs_summaries* sobre estas predicciones en el caso de variables continuas.
5. *Predictions* y con la ayuda de la función *gs_summaries*, evaluamos la capacidad predictiva del modelo entrenado para el tipo de kernel especificado. La función *gs_summaries* devuelve una lista que identificamos como *summaries* y que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.
 6. Finalmente, se crea un elemento con el nombre de kernel especificado en cada una de las listas *PredictionsAll*, *TimesAll*, *HyperparamsAll* y *SummariesAll*, que corresponden a los marcos de datos de las listas *Predictions*, *Times*, *Hyperparams* y *summaries*, respectivamente.

```
for (kernel in kernels) {
  cat("*** Kernel:", kernel, "***\n")
}
```

```

for (line_proportion in lines_proportions) {
  cat("\t*** Line_Proportion:", line_proportion, "***\n")

  # Compute the kernel
  X <- kernelize(
    X,
    kernel = kernel,
    arc_cosine_deep = 2,
    rows_proportion = line_proportion
  )

  # Random Partition
  set.seed(2022)
  folds <- cv_random(
    records_number = nrow(X),
    folds_number = 5,
    testing_proportion = 0.2
  )

  # Empty data frames that will contain Predictions, Times
  # of execution & Summaries for each partition
  Predictions <- data.frame()
  Times <- data.frame()
  Hyperparams <- data.frame()

  for (i in seq_along(folds)) {
    cat("\t*** Fold:", i, "***\n")
    fold <- folds[[i]]

    # Identify the training and testing sets
    X_training <- X[fold$training, ]
    X_testing <- X[fold$testing, ]
    y_training <- y[fold$training]
    y_testing <- y[fold$testing]

    # Model training
    model <- generalized_linear_model(
      x = X_training,
      y = y_training,

      # Specify the hyperparameters values
      alpha = list(min = 0, max = 1),
      lambdas_number = 100,
      tune_folds_number = 5,
      tune_type = "bayesian_optimization",
      tune_bayes_samples_number = 5,
      tune_bayes_iterations_number = 5,
      tune_grid_proportion = 0.8,

```

```

    # In this example the iterations wont be shown
    verbose = FALSE
  )

  # Testing Predictions
  predictions <- predict(model, X_testing)

  # Predictions for the Fold Fold
  Predictions <- data.frame(
    Fold = i,
    Line = PhenoToy$Line[fold$testing],
    Env = PhenoToy$Env[fold$testing],
    Observed = y_testing,
    Predicted = predictions$predicted
  )
  Predictions <- rbind(Predictions, FoldPredictions)

  # Execution times
  FoldTime <- data.frame(
    kernel = kernel,
    Fold = i,
    Minutes = as.numeric(model$execution_time, units = "mins")
  )
  Times <- rbind(Times, FoldTime)

  # Hyperparams for the Fold
  HyperparamsFold <- model$hyperparams_grid %>%
    mutate(Fold = i)
  Hyperparams <- rbind(Hyperparams, HyperparamsFold)
}

# Summaries of the Folds
summaries <- gs_summaries(Predictions)
str_line <- paste("Line_Proportion:", line_proportion)

# Predictions, Times of execution & Summaries for the
# specified Kernel
PredictionsAll[[kernel]][[str_line]] <- Predictions
TimesAll[[kernel]][[str_line]] <- Times
HyperparamsAll[[kernel]][[str_line]] <- Hyperparams
SummariesAll[[kernel]][[str_line]] <- summaries
}
}

#> *** Kernel: Sparse_Gaussian ***
#> *** Line_Proportion: 0.5 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***

```



```

#> *** Line_Proportion: 0.6 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.7 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.8 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.9 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 1 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: Sparse_Arc_cosine ***
#> *** Line_Proportion: 0.5 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.6 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.7 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.8 ***

```

```
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.9 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 1 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
```

Recordando que este proceso se realizó para cada combinación de tipo de kernel y relación de línea especificada, cada una de las listas *PredictionsAll*, *TimesAll*, *HyperparamsAll* y *SummariesAll* contiene las predicciones, los tiempos de ejecución, las combinaciones de hiperparámetros (en este caso *alpha*) y los resúmenes, respectivamente, para cada combinación entre el tipo de kernel y la proporción de líneas aplicadas a la matriz de datos X. A modo de ejemplo, a continuación se muestran los resultados obtenidos para el tipo de kernel “Sparse_Gaussian” y “Line_Proportion: 0.7”:

```
# Predictions for the Sparse_Gaussian Kernel
head(PredictionsAll$Sparse_Gaussian$`Line_Proportion: 0.7`)
#>   Fold   Line          Env Observed Predicted
#> 1     5 ICGV05057 ALIYARNAGAR_R15    13.06  9.236051
#> 2     5 ICG2857   ICRISAT_R15     4.96  9.456688
#> 3     5   TG19   ICRISAT_R15     7.66  9.901224
#> 4     5   DTG3   JALGOAN_R15    12.00  8.926199
#> 5     5   TG19 ICRISAT_PR15-16     4.33  9.022342
#> 6     5 ICGV95377   JALGOAN_R15    10.30  8.844390

# Times of execution for the Sparse_Gaussian Kernel
TimesAll$Sparse_Gaussian$`Line_Proportion: 0.7`
#>      kernel Fold   Minutes
#> 1 Sparse_Gaussian    1 0.06758389
#> 2 Sparse_Gaussian    2 0.08593390
#> 3 Sparse_Gaussian    3 0.06968764
#> 4 Sparse_Gaussian    4 0.08042319
#> 5 Sparse_Gaussian    5 0.09610434

# Elements of SummariesAll
names(SummariesAll)
#> [1] "Sparse_Gaussian" "Sparse_Arc_cosine"
```

```

# Elements of summaries for Sparse_Gaussian Kernel
names(SummariesAll$Sparse_Gaussian)
#> [1] "Line_Proprtion: 0.5" "Line_Proprtion: 0.6"
#> [3] "Line_Proprtion: 0.7" "Line_Proprtion: 0.8"
#> [5] "Line_Proprtion: 0.9" "Line_Proprtion: 1"
names(SummariesAll$Sparse_Gaussian$`Line_Proprtion: 0.7`)
#> [1] "line" "env" "fold"

# Summaries by Line
head(SummariesAll$Sparse_Gaussian$`Line_Proprtion: 0.7`$line)
#>      Line Observed Predicted Difference
#> 1 ICGV00248   9.0000    9.1482    0.1482
#> 2 ICGV95377   9.2100    9.5088    0.2988
#> 3 ICG9315     8.7000    9.1658    0.4658
#> 4 ICGV00362   8.3100    9.6635    1.3535
#> 5 ICCV04107 190.6667 192.3172    1.6505
#> 6 ICG2857     7.5450    9.2703    1.7253

# Summaries by Enviroment
SummariesAll$Sparse_Gaussian$`Line_Proprtion: 0.7`$env[, 1:8]
#>      Env      MSE MSE_SE   RMSE RMSE_SE  NRMSE
#> 1 ALIYARNAGAR_R15   4.2365    NA  2.0583    NA 1.0617
#> 2 ICRISAT_PR15-16 19.3393    NA  4.3976    NA 3.1948
#> 3 ICRISAT_R15     14.5566    NA  3.8153    NA 0.9579
#> 4 JALGOAN_R15     49.6871    NA  7.0489    NA 1.0946
#> 5      1 8663.5554    NA 93.0782    NA 0.9584
#> 6      2 4029.6301    NA 63.4794    NA 0.8882
#> 7      4 1338.8986    NA 36.5910    NA 1.0354
#> 8      5  510.6866    NA 22.5984    NA 1.1395
#> 9      6 3213.0426    NA 56.6837    NA 1.1018
#> 10     7  979.0036    NA 31.2890    NA 0.8486
#> 11 Global 1774.9271    NA 42.1299    NA 0.3176
#>      NRMSE_SE      MAE
#> 1      NA  1.5401
#> 2      NA  4.1352
#> 3      NA  3.5174
#> 4      NA  5.4254
#> 5      NA 76.5885
#> 6      NA 60.1064
#> 7      NA 32.0289
#> 8      NA 20.4767
#> 9      NA 43.7637
#> 10     NA 25.6632
#> 11     NA 26.8566
SummariesAll$Sparse_Gaussian$`Line_Proprtion: 0.7`$env[, 9:15]
#>      MAE_SE      Cor Cor_SE Intercept Intercept_SE      Slope
#> 1      NA -0.2825    NA   23.4790    NA -1.4375
#> 2      NA -0.5636    NA   18.0558    NA -1.4766
#> 3      NA -0.2218    NA   23.5443    NA -1.4194
#> 4      NA  0.3115    NA  -37.5172    NA  5.5495

```

```

#> 5      NA 0.1755      NA -828.8236      NA 4.2187
#> 6      NA -0.9770     NA 5318.6931     NA -14.3785
#> 7      NA 0.6586      NA -1161.1830    NA 6.0003
#> 8      NA -0.4061     NA 748.5099     NA -1.8713
#> 9      NA -0.1788     NA 3585.0368    NA -9.2182
#> 10     NA 0.9032      NA -387.2714    NA 2.8679
#> 11     NA 0.9488      NA 1.2152      NA 0.9565
#>      Slope_SE
#> 1      NA
#> 2      NA
#> 3      NA
#> 4      NA
#> 5      NA
#> 6      NA
#> 7      NA
#> 8      NA
#> 9      NA
#> 10     NA
#> 11     NA
SummariesAll$Sparse_Gaussian$`Line_Proprtion: 0.7`$env[, 16:19]
#>      R2 R2_SE  MAAPE  MAAPE_SE
#> 1 0.0798    NA 0.1381    NA
#> 2 0.3177    NA 0.7051    NA
#> 3 0.0492    NA 0.3612    NA
#> 4 0.0970    NA 0.3760    NA
#> 5 0.0308    NA 0.3060    NA
#> 6 0.9544    NA 0.1824    NA
#> 7 0.4337    NA 0.1173    NA
#> 8 0.1649    NA 0.0755    NA
#> 9 0.0320    NA 0.1494    NA
#> 10 0.8158    NA 0.1622    NA
#> 11 0.9003    NA 0.2308    NA

# Summaries by Fold
SummariesAll$Sparse_Gaussian$`Line_Proprtion: 0.7`$fold[, 1:8]
#>      Fold      MSE  MSE_SE  RMSE RMSE_SE  NRMSE NRMSE_SE
#> 1      5 1882.264 876.892 32.1040 9.7274 1.2281 0.2206
#> 2 Global 1774.927      NA 42.1299      NA 0.3176      NA
#>      MAE
#> 1 27.3245
#> 2 26.8566
SummariesAll$Sparse_Gaussian$`Line_Proprtion: 0.7`$fold[, 9:15]
#>      MAE_SE  Cor Cor_SE Intercept Intercept_SE  Slope
#> 1 8.2277 -0.0581 0.1808 730.2524 654.5285 -1.1165
#> 2      NA 0.9488      NA 1.2152      NA 0.9565
#>      Slope_SE
#> 1 2.0598
#> 2      NA
SummariesAll$Sparse_Gaussian$`Line_Proprtion: 0.7`$fold[, 16:19]
#>      R2 R2_SE  MAAPE  MAAPE_SE

```

```
#> 1 0.2975 0.1069 0.2573 0.0597
#> 2 0.9003 NA 0.2308 NA
```

Además, los elementos de la lista *HyperparamsAll* contienen las columnas *alpha*, *loss** y *Fold*, donde el valor de la columna de pérdida corresponde al costo del modelo para cada combinación de los α valores de partición y, ordenados de menor a mayor dentro de cada partición.

```
# First rows of Hyperparams
head(HyperparamsAll$Sparse_Gaussian$`Line_Proprtion: 0.7`)
#>      alpha      mse Fold
#> 5 0.07803758 18.92680 1
#> 10 0.11056155 18.96947 1
#> 4 0.16433226 19.03684 1
#> 2 0.42398546 19.14971 1
#> 8 0.52101372 19.16375 1
#> 9 0.54698581 19.16473 1

# Last rows of Hyperparams
tail(HyperparamsAll$Sparse_Gaussian$`Line_Proprtion: 0.7`)
#>      alpha      mse Fold
#> 74 0.45703866 16.78341 5
#> 44 0.62443722 16.79359 5
#> 64 0.76445986 16.80660 5
#> 54 0.92873054 16.82231 5
#> 34 0.14337780 16.88426 5
#> 24 0.05022715 16.96390 5
```

4 Métodos de regresión bayesiana

4.1 Ejemplo de resultados continuos con Bayesian Lasso con solo *G* en el predictor con Grid search y particiones aleatorias

Este ejemplo evalúa un modelo LASSO bayesiano con cinco particiones aleatorias, con 20% los datos para el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de cada partición (los parámetros predeterminados de la función *cv_random*), para una respuesta continua, usando solo la matriz *G* (Matriz de diseño de línea que contiene información genómica) como predictor.

En este ejemplo, el conjunto de datos utilizado es *GroundnutToy* y el objetivo es predecir la variable continua *YPH* del data frame *PhenoToy* utilizando la matriz *G* descrita anteriormente como predictor; por lo que identificamos las variables predictoras y de respuesta como *X* y *y* respectivamente.

```
# Load the data
load("GroundnutToy.RData", verbose = TRUE)
#> Loading objects:
#> PhenoToy
```

```

#>   GenoToy
# Data preparation of G
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %**% Geno

# Predictor and Response Variables
X <- LineG
y <- PhenoToy$YPH

# Note that y is a continuous numeric vector
class(y)
#> [1] "numeric"
typeof(y)
#> [1] "double"

```

Posteriormente, realizamos cinco particiones aleatorias, con 80% los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función `cv_random` (con los parámetros por defecto). Además, creamos el data frame `Predictions` vacío que servirá para almacenar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función `gs_summaries`.

```

# Set seed for reproducible results
set.seed(2022)
folds <- cv_random(records_number = nrow(X))

# A data frame that will contain the variables:
Predictions <- data.frame()

```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifica el conjunto de prueba de la variable de respuesta. Tenga en cuenta que, a diferencia de los modelos lineales generalizados, ahora solo es necesario identificar el conjunto de prueba, ya que la función `bayesian_model` tiene un argumento `testing_indices` que corresponde a los índices de este conjunto;
2. El modelo se entrena con el conjunto de entrenamiento, indicando con la función `bayesian_model` una lista que especifica la matriz de predictores y el modelo, además de la variable respuesta y los índices correspondientes al conjunto de prueba;
3. Con el modelo obtenido en (2), se predice la variable respuesta `YPH` en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. `FoldPredictions` que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba.

- b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*.

```
# Model training and predictions of the ith partition
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")
  fold <- folds[[i]]

  # Identify the testing response set
  y_testing <- y[fold$testing]

  # Model training with Bayesian LASSO Regression
  ETA <- list(G = list(x = X, model = "Bayes_Lasso"))
  model <- bayesian_model(
    x = ETA,
    y = y,
    testing_indices = fold$testing
  )

  # Prediction of the test set
  predictions <- predict(model, fold$testing)

  # Predictions for the Fold Fold
  FoldPredictions <- data.frame(
    Fold = i,
    Line = PhenoToy$Line[fold$testing],
    Env = PhenoToy$Env[fold$testing],
    Observed = y_testing,
    Predicted = predictions$predicted
  )
  Predictions <- rbind(Predictions, FoldPredictions)
}
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1615 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1421 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1463 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1365 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.142 secs ***
```

Predictions data frame contiene las columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición, correspondiente al formato

necesario para usar la función *gs_summaries* en estas predicciones en el caso de variables continuas.

La función *gs_summaries* devuelve una lista que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```
head(Predictions)
#>   Fold      Line      Env Observed Predicted
#> 1     1 ICGV97115  JALGOAN_R15    817.85  1745.308
#> 2     1 ICG9315   ICRISAT_R15   1324.07  1452.833
#> 3     1 ICGV06099 ICRISAT_PR15-16 2334.15  2210.811
#> 4     1 ICGV00248   ICRISAT_R15   1993.36  1756.494
#> 5     1 ICGV05057   ICRISAT_R15   1856.64  1754.378
#> 6     1 ICGV02434  JALGOAN_R15    367.32  1648.263
unique(Predictions$Fold)
#> [1] 1 2 3 4 5

# Summaries
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted Difference
#> 1 ICGV07217 1530.160  1509.888    20.2720
#> 2  ICG9315 1453.340  1427.694    25.6456
#> 3 Gangapuri 1232.337  1154.666    77.6712
#> 4      TG19 1106.877  1197.769    90.8921
#> 5  ICG15419 1307.694  1429.795   122.1015
#> 6 ICGV99085 1398.842  1262.463   136.3792

# Summaries by Environment
summaries$env[, 1:7]
#>      Env      MSE      MSE_SE      RMSE      RMSE_SE      NRMSE
#> 1 ALIYARNAGAR_R15 225579.7  82778.39 435.3207  94.9678  1.4216
#> 2 ICRISAT_PR15-16 455330.4 211762.76 620.4828 132.6004  1.1534
#> 3  ICRISAT_R15 177927.4  32573.06 411.8567  45.5560  0.7422
#> 4  JALGOAN_R15 801035.1 207070.53 859.6842 124.4770  0.9782
#> 5      Global 416059.0 108383.49 623.9185  81.8301  0.8473
#>      NRMSE_SE
#> 1    0.8773
#> 2    0.1695
#> 3    0.1006
#> 4    0.1010
#> 5    0.0295
summaries$env[, 8:14]
#>      MAE      MAE_SE      Cor      Cor_SE Intercept Intercept_SE
#> 1 367.3761  89.5063 0.8264 0.0966 -434.7518    176.0840
```



```

#> 2 482.5078 107.3539 0.2623 0.1449 735.7532 634.0835
#> 3 362.8939 41.2606 0.6939 0.0711 -278.5361 242.3355
#> 4 697.3011 96.5122 0.2369 0.3209 -30.3277 1663.8286
#> 5 480.6238 49.8928 0.5795 0.0707 -392.2724 565.8855
#> Slope
#> 1 1.1194
#> 2 0.6344
#> 3 1.1708
#> 4 1.0550
#> 5 1.3228
summaries$env[, 15:19]
#> Slope_SE R2 R2_SE MAAPE MAAPE_SE
#> 1 0.1958 0.7203 0.1358 0.3338 0.0960
#> 2 0.4799 0.1527 0.0912 0.3003 0.0480
#> 3 0.1943 0.5018 0.1123 0.2560 0.0350
#> 4 1.1659 0.4679 0.1078 0.3887 0.0606
#> 5 0.4161 0.3559 0.0764 0.2991 0.0130

# Summaries by Fold
summaries$fold[, 1:8]
#> Fold MSE MSE_SE RMSE RMSE_SE NRMSE NRMSE_SE
#> 1 1 569460.2 279247.13 670.6226 199.7712 0.8002 0.1661
#> 2 2 427782.7 277634.38 554.1137 200.6163 0.6719 0.1487
#> 3 3 338101.3 66028.75 573.7242 54.5950 2.0501 0.9793
#> 4 4 196998.0 50168.44 431.8725 59.1160 0.8882 0.1828
#> 5 5 542498.4 259529.17 678.8476 164.9893 0.9588 0.1113
#> 6 Global 416059.0 108383.49 623.9185 81.8301 0.8473 0.0295
#> MAE
#> 1 555.8062
#> 2 447.0548
#> 3 481.0394
#> 4 360.8776
#> 5 542.8207
#> 6 480.6238

```

4.2 Ejemplo de con una respuesta binaria con Bayes A $Env + G$ en el predictor con Grid search y validación cruzada 7-fold

Este ejemplo evalúa un modelo baesiano A bayesiano con validación cruzada 7-fold, para una respuesta binaria, utilizando el efecto del ambiente y la matriz G como predictores.

En este ejemplo, el conjunto de datos utilizado es *EYTToy* y el objetivo es predecir la variable binaria y_{bin} , que es una transformación de la variable *Biomasa* que indica si la respuesta es mayor que la mediana de esta variable o no, utilizando la matriz de diseño de *PhenoToy*. Env variable y matrix G , descritos anteriormente, como predictores; por lo que identificamos las variables predictoras y de respuesta como X y y_{bin} respectivamente.

```

# Load the data
load("EYTToy.RData", verbose = TRUE)
#> Loading objects:

```

```
#> PhenoToy
#> GenoToy
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %%% Geno

X <- cbind(Env, LineG)
y_bin <- BurStMisc::ntile(PhenoToy$Height, 2, result = "factor")
#> Warning in BurStMisc::ntile(PhenoToy$Height, 2, result =
#> "factor"): common values across groups: 1, 2
```

Tenga en cuenta que la variable de respuesta y_{bin} es un factor con solo dos niveles (o categorías), lo cual es importante para que el modelo se entrene automáticamente para una variable binaria (regresión logística). Por esta razón, es importante tener en cuenta esas variables de respuesta binarias o categóricas antes de usar la función *bayesian_model*.

Posteriormente hacemos las particiones correspondientes a 7-Fold CV, con la ayuda de la función *cv_kfold*. Además, creamos el data frame vacío *Predictions* que se utilizará para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
# Set seed for reproducible results
set.seed(2022)
folds <- cv_kfold(records_number = nrow(X), k = 7)

# A data frame that will contain the variables:
Predictions <- data.frame()
```

Posteriormente, se seguirá el siguiente proceso **para cada artición**:

1. Se identifica el conjunto de prueba de la variable de respuesta;
2. El modelo se entrena con el conjunto de entrenamiento, indicando con la función *bayesian_model* una lista que especifica la matriz de predictores y el modelo, además de la variable respuesta y los índices correspondientes al conjunto de prueba;
3. Con el modelo obtenido en (2), y_{bin} se predice la variable respuesta en el conjunto de prueba, con el fin de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: *Fold* number, *Line*, *Env*, *Observed*, *Predicted*, 1 y 2 para cada elemento del conjunto de prueba. Tenga en cuenta que, a diferencia del ejemplo anterior, ahora tenemos dos columnas adicionales correspondientes a las probabilidades asociadas con cada elemento correspondiente a esa categoría.

- b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*. Con esto, *Predictions* se formatea para que sea un argumento de la función *gs_summaries*.

```
# Model training and predictions of the ith partition
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")
  fold <- folds[[i]]

  # Identify the testing response set
  y_testing <- y_bin[fold$testing]

  # Model training with Bayes A model
  ETA <- list(list(x = X, model = "Bayes_A"))
  model <- bayesian_model(
    x = ETA,
    y = y_bin,
    testing_indices = fold$testing
  )

  # Prediction of the test set
  predictions <- predict(model, fold$testing)

  # Predictions for the Fold Fold
  FoldPredictions <- cbind(
    data.frame(
      Fold = i,
      Line = PhenoToy$Line[fold$testing],
      Env = PhenoToy$Env[fold$testing],
      Observed = y_testing,
      Predicted = predictions$predicted
    ),
    predictions$probabilities
  )
  Predictions <- rbind(Predictions, FoldPredictions)
}

#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.2304 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.2292 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.2058 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.2197 secs ***
#> *** Fold: 5 ***
```

```
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.2342 secs ***
#> *** Fold: 6 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.2306 secs ***
#> *** Fold: 7 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.2391 secs ***
```

Predictions contiene columnas *Fold*, *Line*, *Env*, *Observed*, *Predicted*, *A* y *B* para cada lemento del conjunto de prueba de cada artición, correspondiente al formato necesario para usar la función *gs_summaries* en *Prediction* en el caso de variables binarias.

La función *gs_summaries* devuelve una lista que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```
head(Predictions)
#>   Fold      Line      Env Observed Predicted      1
#> 1     1 GID7625106 Flat5IR      1         1 0.68447526
#> 2     1 GID7625276 FlatDrip      1         1 0.99846345
#> 3     1 GID7625985 Bed5IR      2         2 0.08935119
#> 4     1 GID7626366      EHT      1         2 0.40729747
#> 5     1 GID7626381 FlatDrip      1         1 0.99382174
#> 6     1 GID7626446 FlatDrip      1         1 0.95210362
#>           2
#> 1 0.315524738
#> 2 0.001536546
#> 3 0.910648806
#> 4 0.592702529
#> 5 0.006178261
#> 6 0.047896383
unique(Predictions$Fold)
#> [1] 1 2 3 4 5 6 7
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"

# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted      X1      X2
#> 1 GID7462121      2         2 0.3697 0.6303
#> 2 GID7625106      1         1 0.6985 0.3015
#> 3 GID7625276      1         1 0.6786 0.3214
#> 4 GID7625985      2         2 0.3488 0.6512
#> 5 GID7626366      1         2 0.5448 0.4552
#> 6 GID7626381      1         1 0.6439 0.3561

# Summaries by Environment
```

```

summaries$env
#>      Env  PCCC PCCC_SE  Kappa Kappa_SE BrierScore
#> 1  Bed5IR 0.8071  0.0579  0.1250  0.1664    0.2364
#> 2    EHT 0.7878  0.0928  0.4520  0.1860    0.3877
#> 3 Flat5IR 0.5534  0.0704 -0.0197  0.1416    0.5193
#> 4 FlatDrip 1.0000  0.0000    NaN    NA    0.0022
#> 5  Global 0.8021  0.0276  0.5930  0.0563    0.2912
#>      BrierScore_SE
#> 1      0.0464
#> 2      0.0381
#> 3      0.0724
#> 4      0.0008
#> 5      0.0345

# Summaries by Fold
summaries$fold
#>      Fold  PCCC PCCC_SE  Kappa Kappa_SE BrierScore
#> 1      1 0.8250  0.1181  0.2727  0.1928    0.2033
#> 2      2 0.7750  0.1315  0.1389  0.3735    0.3476
#> 3      3 0.6577  0.1546 -0.2222  0.1273    0.3616
#> 4      4 0.6583  0.1493  0.0000  0.0000    0.3032
#> 5      5 0.8542  0.0859  0.3333  0.2887    0.2392
#> 6      6 0.8167  0.1067  0.4444  0.2546    0.3328
#> 7      7 0.9226  0.0449  0.3478  0.2460    0.2171
#> 8 Global 0.8021  0.0276  0.5930  0.0563    0.2912
#>      BrierScore_SE
#> 1      0.1002
#> 2      0.1487
#> 3      0.1708
#> 4      0.1036
#> 5      0.0801
#> 6      0.1527
#> 7      0.0823
#> 8      0.0345

```

4.3 Ejemplo con variable respuesta categórica con Bayes C con optimización bayesiana con particiones aleatorias *Env* + *G* + *GE* dentro del predictor

Este ejemplo evalúa un modelo Bayesiano (Bayes C) con cinco particiones aleatorias, con 20% los datos para el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de cada partición (los parámetros predeterminados de la función `cv_random`), para una respuesta categórica, usando el efecto de Environment, la matriz *G* y la interacción entre estos dos como predictores.

En este ejemplo, el conjunto de datos utilizado es *ChickpeaToy* y buscamos predecir la variable categórica *y*, que es una transformación de la variable *Biomass del marco de datos PhenoToy* usando la función `ntile`, usando la matriz de diseño de la variable *Env* de *PhenoToy*, la matriz *G* descrito anteriormente y la matriz de diseño de la interacción entre

estos dos, como predictores; por lo que identificamos las variables predictoras y de respuesta como X y y respectivamente.

```
# Load the data
load("ChickpeaToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env, G & GE
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First Column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %%% Geno
LineGenoEnv <- model.matrix(~ 0 + LineG:Env)

# Predictor and Response Variables
X <- cbind(Env, LineG, LineGenoEnv)
y <- BurStMisc::ntile(PhenoToy$Biomass, 3, result = "factor")
print(y[1:30])
#> [1] 2 3 2 3 3 1 1 2 1 2 3 1 2 3 2 2 3 2 3 3 3 2 3 1 2 3 2 2 3
#> [30] 2
#> Levels: 1 < 2 < 3
```

Tenga en cuenta que la variable de respuesta y es un factor con tres niveles (o categorías), lo cual es importante para que el modelo se entrene automáticamente para una variable ategórica (**modelo multinomial simétrico**). Por esta razón, es importante tener en cuenta esas variables de respuesta binarias o categóricas antes de usar la función *bayesian_model*.

Posteriormente, realizamos cinco particiones aleatorias, con 80% los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función *cv_random* (con los parámetros por defecto). Además, creamos el data frame vacío *Predictions* que se utilizará para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
# Set seed for reproducible results
set.seed(2022)
folds <- cv_random(records_number = nrow(X))

# A data frame that will contain the variables:
Predictions <- data.frame()
```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifica el conjunto de prueba de la variable de respuesta;

2. El modelo se entrena con el conjunto de entrenamiento, indicando con la función *bayesian_model* una lista que especifica la matriz de predictores y el modelo, además de la variable respuesta y los índices correspondientes al conjunto de prueba;
3. Con el modelo obtenido en (2), se predice la variable respuesta y en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed*, *Predicted*, 1, 2 y 3 para cada elemento del conjunto de prueba. Tenga en cuenta que, a diferencia de los ejemplos anteriores, ahora tenemos tres columnas adicionales correspondientes a las probabilidades asociadas con cada elemento correspondiente a esa categoría.
 - b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*. Con esto, *Predictions* se formatea para que sea un argumento de la función *gs_summaries*.

```
# Model training and predictions of the ith partition
```

```
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")
  fold <- folds[[i]]

  # Identify the testing response set
  y_testing <- y[fold$testing]

  # Model training with Bayesian C Regression
  ETA <- list(G = list(x = X, model = "Bayes_C"))
  model <- bayesian_model(
    x = ETA,
    y = y,
    testing_indices = fold$testing
  )

  # Prediction of the test set
  predictions <- predict(model, fold$testing)

  # Predictions for the Fold Fold
  FoldPredictions <- cbind(
    data.frame(
      Fold = i,
      Line = PhenoToy$Line[fold$testing],
      Env = PhenoToy$Env[fold$testing],
      Observed = y_testing,
      Predicted = predictions$predicted
    ),
    predictions$probabilities
  )
}
```

```

)
Predictions <- rbind(Predictions, FoldPredictions)
}
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.4127 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.4172 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.4188 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.4578 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.4204 secs ***

```

Predictions contiene las columnas *Fold*, *Line*, *Env*, *Observed*, *Predicted*, 1, 2* y 3 para cada elemento del conjunto de prueba de cada partición, correspondiente al formato necesario para usar la función *gs_summaries* en *Prediction* en el caso de variables categóricas.

La función *gs_summaries* devuelve una lista que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```

head(Predictions)
#>   Fold      Line Env Observed Predicted      1      2
#> 1     1 ICCV97301   6         3         3 0.008826404 0.2037639
#> 2     1 ICCV04103   1         3         2 0.295791234 0.5512370
#> 3     1 ICCV05109   4         2         2 0.388952599 0.5223319
#> 4     1 ICCV00402   7         1         1 0.773580463 0.2133879
#> 5     1 ICCV09114   4         2         2 0.446518625 0.4760298
#> 6     1 ICCV03102   2         3         3 0.008588422 0.1918917
#>           3
#> 1 0.78740968
#> 2 0.15297175
#> 3 0.08871551
#> 4 0.01303164
#> 5 0.07745156
#> 6 0.79951992
unique(Predictions$Fold)
#> [1] 1 2 3 4 5
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env"  "fold"
# Summaries by Line
head(summaries$line)

```



```
#>      Line Observed Predicted      X1      X2      X3
#> 1 ICCV00402         2         2 0.4020 0.3745 0.2235
#> 2 ICCV01301         1         1 0.4167 0.3340 0.2493
#> 3 ICCV03102         2         2 0.2727 0.3857 0.3416
#> 4 ICCV03104         3         2 0.1900 0.3438 0.4663
#> 5 ICCV03105         3         3 0.0771 0.2507 0.6723
#> 6 ICCV03107         3         2 0.1229 0.3721 0.5050
```

Summaries by Environment

```
summaries$env
```

```
#>      Env      PCCC PCCC_SE      Kappa Kappa_SE BrierScore
#> 1      1 0.4342 0.0695 0.1047 0.0814 0.6492
#> 2      2 0.6253 0.0697 -0.0075 0.0075 0.4341
#> 3      4 0.6131 0.1093 0.3800 0.1471 0.5560
#> 4      5 0.4889 0.1616 0.0000 0.0000 0.6040
#> 5      6 0.8533 0.0904 0.0000 0.0000 0.2823
#> 6      7 0.9600 0.0400 0.0000      NA 0.1003
#> 7 Global 0.6052 0.0570 0.3927 0.0893 0.5042
#>      BrierScore_SE
#> 1      0.0369
#> 2      0.0544
#> 3      0.0408
#> 4      0.0974
#> 5      0.0794
#> 6      0.0469
#> 7      0.0394
```

Summaries by Fold

```
summaries$fold
```

```
#>      Fold      PCCC PCCC_SE      Kappa Kappa_SE BrierScore
#> 1      1 0.6283 0.1718 0.1668 0.1453 0.4753
#> 2      2 0.7695 0.1035 0.0708 0.0638 0.3749
#> 3      3 0.7421 0.1169 0.1875 0.1531 0.3638
#> 4      4 0.5310 0.1033 0.0305 0.0279 0.4994
#> 5      5 0.6415 0.0403 0.1006 0.0689 0.4749
#> 6 Global 0.6052 0.0570 0.3927 0.0893 0.5042
#>      BrierScore_SE
#> 1      0.1416
#> 2      0.0785
#> 3      0.1110
#> 4      0.0965
#> 5      0.0483
#> 6      0.0394
```

4.4 Ejemplo de resultado continuo con GBLUP con optimización bayesiana con línea de partición aleatoria *Env* + *G* + *GE* dentro del predictor

Este ejemplo evalúa un modelo bayesiano BGBLUP con cinco particiones aleatorias del conjunto de líneas, con 20% líneas para el conjunto de prueba y 80% para el conjunto de

entrenamiento dentro de cada partición, para una respuesta continua, utilizando el efecto del ambiente, la matriz G y la interacción entre estos dos como predictores.

En este ejemplo, el dataset utilizado es *MaizeToy* y busca predecir la variable continua *Yield*, utilizando como predictores la matriz de diseño de la *variable Env* de *PhenoToy*, la matriz G descrita anteriormente y la matriz de diseño de la interacción entre estas dos; por lo que identificamos las variables predictoras y de respuesta como X y y respectivamente.

```
# Load the data
load("MaizeToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy
# Data preparation of Env, G & GE
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %*% Geno
LinexGenoxEnv <- model.matrix(~ 0 + LineG:Env)

# Predictor and Response Variables
X <- cbind(Env, LineG, LinexGenoxEnv)
K <- X %*% t(X) # Linear Kernel
y <- PhenoToy$Yield
print(y[1:15])
#> [1] 6.11 6.21 5.32 6.62 5.60 6.24 5.24 4.93 6.70 4.72 4.46
#> [12] 6.44 4.98 5.84 6.97
typeof(y)
#> [1] "double"
```

Posteriormente, realizamos cinco particiones aleatorias del conjunto de líneas, con 80% este conjunto para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función `cv_random` (con los parámetros por defecto). Además, creamos el marco de datos vacío `Predictions` que se utilizará para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función `gs_summaries`.

```
# Set seed for reproducible results
set.seed(2022)
# Unique Lines
GIDs <- unique(PhenoToy$Line)
folds <- cv_random(length(GIDs))

# A data frame that will contain the variables:
Predictions <- data.frame()
```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifica el conjunto de prueba de las variables de respuesta, identificando primero las líneas correspondientes a este conjunto;
2. El modelo se entrena con el conjunto de entrenamiento, indicando con la función *bayesian_model* una lista que especifica la matriz de predictores y el modelo, además de la variable respuesta y los índices correspondientes al conjunto de prueba;
3. Con el modelo obtenido en (2), se predice la variable de respuesta *Rendimiento* en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba.
 - b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*. Con esto, *Predictions* se formatea para que sea un argumento de la función *gs_summaries*.

```
# Model training and predictions of the ith partition
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")
  # Identify the training and testing Line sets
  fold <- folds[[i]]
  Lines_tst_i <- GIDs[fold$testing]
  tst_i <- which(PhenoToy$Line %in% Lines_tst_i)

  # Identify the testing se
  y_testing <- y[tst_i]

  # Model training with BGBLUP model
  ETA <- list(list(x = K, model = "BGBLUP"))
  model <- bayesian_model(
    x = ETA,
    y = y,
    testing_indices = tst_i
  )

  # Prediction of the test set
  predictions <- predict(model, tst_i)

  # Predictions for the Fold Fold
  FoldPredictions <- data.frame(
    Fold = i,
    Line = PhenoToy$Line[tst_i],
    Env = PhenoToy$Env[tst_i],
    Observed = y_testing,
    Predicted = predictions$predicted
  )
}
```

```

)
Predictions <- rbind(Predictions, FoldPredictions)
}
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1047 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1037 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1185 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.117 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1054 secs ***

```

Predictions contiene columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición, correspondiente al formato necesario para usar la función *gs_summaries en Prediction* en el caso de contar variabes.

La función *gs_summaries* devuelve una lista que contiee tres marcos de daos correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```

head(Predictions)
#>   Fold      Line Env Observed Predicted
#> 1     1 CKDHL0049 EBU    4.72  6.371418
#> 2     1 CKDHL0049 KAK    4.46  5.542784
#> 3     1 CKDHL0049 KTI    6.44  6.174760
#> 4     1 CKDHL0108 EBU    7.47  6.523753
#> 5     1 CKDHL0108 KAK    5.67  5.186489
#> 6     1 CKDHL0108 KTI    5.25  5.988327
unique(Predictions$Fold)
#> [1] 1 2 3 4 5
# Summaries
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted Difference
#> 1 CKDHL0515  6.0100    5.9794    0.0306
#> 2 CKDHL0027  5.8800    5.7562    0.1238
#> 3 CKDHL0054  5.9233    5.7864    0.1369
#> 4 CKDHL0160  5.7633    5.9365    0.1732
#> 5 CKDHL0150  5.7800    5.9749    0.1949

```

```

#> 6 CKDHL0529    5.8067    6.0261    0.2195

# Summaries by Environment
summaries$env[, 1:8]
#>      Env      MSE MSE_SE  RMSE RMSE_SE  NRMSE NRMSE_SE  MAE
#> 1   EBU 0.5385 0.1175 0.7128 0.0873 1.0584 0.0787 0.5722
#> 2   KAK 0.7519 0.1588 0.8419 0.1038 1.1273 0.1408 0.6559
#> 3   KTI 1.4316 0.3315 1.1621 0.1423 1.0214 0.0767 0.9948
#> 4 Global 0.5320 0.0999 0.7158 0.0699 1.1017 0.0623 0.5428

summaries$env[, 9:15]
#>      MAE_SE      Cor Cor_SE Intercept Intercept_SE  Slope
#> 1 0.0759 0.1570 0.1646 -5.0537 10.8883 1.7878
#> 2 0.0823 0.1598 0.2295 -7.0512 10.5689 2.3269
#> 3 0.1192 -0.0418 0.2327 -1.4818 17.8774 1.2179
#> 4 0.0480 -0.1728 0.1872 10.2343 9.2027 -0.7462
#>      Slope_SE
#> 1 1.6774
#> 2 2.0312
#> 3 2.9025
#> 4 1.5376

summaries$env[, 16:19]
#>      R2 R2_SE  MAAPE MAAPE_SE
#> 1 0.1331 0.1276 0.0911 0.0136
#> 2 0.2362 0.1341 0.1359 0.0229
#> 3 0.2183 0.1041 0.1669 0.0220
#> 4 0.1700 0.0983 0.0962 0.0128
# Summaries by Fold
summaries$fold[, 1:8]
#>      FoLd      MSE MSE_SE  RMSE RMSE_SE  NRMSE NRMSE_SE  MAE
#> 1      1 1.0733 0.1305 1.0319 0.0652 1.0209 0.1309 0.8176
#> 2      2 0.5330 0.2601 0.6892 0.1703 0.9611 0.0377 0.5714
#> 3      3 1.1398 0.6716 0.9633 0.3254 1.2105 0.1289 0.7446
#> 4      4 0.6360 0.0457 0.7964 0.0290 1.1864 0.2136 0.7099
#> 5      5 1.1546 0.3780 1.0472 0.1704 0.9661 0.0341 0.8612
#> 6 Global 0.5320 0.0999 0.7158 0.0699 1.1017 0.0623 0.5428

summaries$fold[, 9:15]
#>      MAE_SE      Cor Cor_SE Intercept Intercept_SE  Slope
#> 1 0.0635 0.4732 0.3078 -29.0643 18.9081 5.4771
#> 2 0.1594 0.0714 0.4183 5.9815 20.3579 0.3570
#> 3 0.2768 -0.1601 0.0882 9.6886 2.5887 -0.6355
#> 4 0.0235 -0.1530 0.1419 13.7142 6.8266 -1.3398
#> 5 0.2024 0.2270 0.1651 -22.9645 15.0740 5.0288
#> 6 0.0480 -0.1728 0.1872 10.2343 9.2027 -0.7462
#>      Slope_SE
#> 1 3.1300
#> 2 3.6097

```

```
#> 3 0.3967
#> 4 1.1393
#> 5 2.5565
#> 6 1.5376

summaries$fold[, 16:19]
#>      R2  R2_SE  MAAPE  MAAPE_SE
#> 1 0.4133 0.1975 0.1663 0.0257
#> 2 0.3551 0.2107 0.0953 0.0270
#> 3 0.0412 0.0352 0.1139 0.0341
#> 4 0.0637 0.0330 0.1152 0.0066
#> 5 0.1060 0.0931 0.1659 0.0408
#> 6 0.1700 0.0983 0.0962 0.0128
```

4.5 Ejemplo de multivariado con respuestas continuas con regresión Bayesian Ridge Con optimización bayesiana con validación cruzada 7-fold con *Env* + *G* el predictor

Este ejemplo evalúa un modelo bayesiano con validación cruzada de 7 veces, para dos respuestas continuas, utilizando el efecto del ambiente y la matriz *G* como predictores.

En este ejemplo, el conjunto de datos utilizado es *GroundnutToy* y el objetivo es predecir las variables continuas *PYPP* y *SYPP* del marco de datos *PhenoToy* utilizando la matriz de diseño de la variable *PhenoToy Env* y la matriz como *G* predictores; por lo que identificamos las variables predictoras y de respuesta como *X* y *y* respectivamente.

```
# Load the data
load("GroundnutToy.RData", verbose = TRUE)
#> Loading objects:
#> PhenoToy
#> GenoToy
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %**% Geno

X <- cbind(Env, LineG)
y <- cbind(PhenoToy$PYPP, PhenoToy$SYPP)
```

Posteriormente hacemos 7 particiones aleatorias, con la ayuda de la función *cv_kfold*. Además, creamos dos data frames vacíos *PredictionsPYPP* y *PredictionsSYPP* que se utilizarán para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
# Set seed for reproducible results
set.seed(2022)
folds <- cv_kfold(records_number = nrow(X), k = 7)
```

```
# Data frames that will contain the variables:
```

```
PredictionsPYPP <- data.frame()
```

```
PredictionsSYPP <- data.frame()
```

Posteriormente, se seguirá el siguiente proceso **paracada partición y para cada variable derespuesta:**

1. Se identifica el conjunto de prueba de la variable de respuesta;
2. El modelo se entrena con el conjunto de entrenamiento, indicando con la función *bayesian_model* una lista que especifica la matriz de predictores y el modelo, además de la variable respuesta y los índices correspondientes al conjunto de prueba;
3. Con el modelo obtenido en (2), se predicen las variables de respuesta *PYPP* y *SYPP* en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de estas variables en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. Se crean los data frames *FoldPredictionsPYPP* y *FoldPredictionSYPP* que contienen las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba y para cada variable de respuesta respectiva.
 - b. Cada fila de *FoldPredictionPYPP* se agrega al data frame *PredictionsPYPP*; y cada fila de *FoldPredictionSYPP* se agrega al data frame *PredictionsSYPP*.

```
# Model training and predictions of the ith partition
```

```
for (i in seq_along(folds)) {  
  cat("*** Fold:", i, "***\n")  
  fold <- folds[[i]]
```

```
  # Identify the testing response set  
  y_testing <- y[fold$testing, ]
```

```
  # Model training with Bayes Ridge Regression
```

```
  ETA <- list(G = list(x = X, model = "BRR"))  
  model <- bayesian_model(  
    x = ETA,  
    y = y,  
    testing_indices = fold$testing  
  )
```

```
  # Prediction of the test set
```

```
  predictions <- predict(model, fold$testing)
```

```
  # Predictions for the ith Fold & PYPP
```

```
  FoldPredictionsPYPP <- data.frame(  
    Fold = i,
```

```

    Line = PhenoToy$Line[fold$testing],
    Env = PhenoToy$Env[fold$testing],
    Observed = y_testing[, 1],
    Predicted = predictions$V1$predicted
  )
  PredictionsPYPP <- rbind(PredictionsPYPP, FoldPredictionsPYPP)

  # Predictions for the ith Fold & SYPP
  FoldPredictionsSYPP <- data.frame(
    Fold = i,
    Line = PhenoToy$Line[fold$testing],
    Env = PhenoToy$Env[fold$testing],
    Observed = y_testing[, 2],
    Predicted = predictions$V2$predicted
  )
  PredictionsSYPP <- rbind(PredictionsSYPP, FoldPredictionsSYPP)
}
#> *** Fold: 1 ***
#> *** Fitting Multivariate Bayesian Model model ***
#> *** Model evaluation completed in 0.9645 secs ***
#> *** Fold: 2 ***
#> *** Fitting Multivariate Bayesian Model model ***
#> *** Model evaluation completed in 0.9975 secs ***
#> *** Fold: 3 ***
#> *** Fitting Multivariate Bayesian Model model ***
#> *** Model evaluation completed in 0.8623 secs ***
#> *** Fold: 4 ***
#> *** Fitting Multivariate Bayesian Model model ***
#> *** Model evaluation completed in 0.859 secs ***
#> *** Fold: 5 ***
#> *** Fitting Multivariate Bayesian Model model ***
#> *** Model evaluation completed in 0.8633 secs ***
#> *** Fold: 6 ***
#> *** Fitting Multivariate Bayesian Model model ***
#> *** Model evaluation completed in 0.8739 secs ***
#> *** Fold: 7 ***
#> *** Fitting Multivariate Bayesian Model model ***
#> *** Model evaluation completed in 0.8714 secs ***

```

Repitiendo este proceso para cada partición, los *data frames* *PredictionsPYPP* y *PredictionSYPP* contienen las columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición en su respectiva variable de respuesta, correspondiente al formato necesario para usar la función *gs_summaries* en estas predicciones en el caso de variables contínuas.

La función *gs_summaries* devuelve una lista que contiene tres *data frames* correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```

head(PredictionsPYPP)
#>   Fold      Line      Env Observed Predicted

```



```

#> 1 1 49x37-99(b)tall ICRISAT_R15 9.24 9.320313
#> 2 1 CSMG84-1 JALGOAN_R15 12.95 9.088764
#> 3 1 DTG15 ALIYARNAGAR_R15 8.09 11.862048
#> 4 1 DTG3 ICRISAT_PR15-16 9.48 6.117894
#> 5 1 Gangapuri JALGOAN_R15 6.10 9.279489
#> 6 1 ICG15419 JALGOAN_R15 10.30 9.343136
unique(PredictionsPYPP$Fold)
#> [1] 1 2 3 4 5 6 7
head(PredictionsSYPP)
#> Fold Line Env Observed Predicted
#> 1 1 49x37-99(b)tall ICRISAT_R15 3.97 5.325267
#> 2 1 CSMG84-1 JALGOAN_R15 8.18 4.933219
#> 3 1 DTG15 ALIYARNAGAR_R15 5.03 7.380684
#> 4 1 DTG3 ICRISAT_PR15-16 5.63 3.905126
#> 5 1 Gangapuri JALGOAN_R15 3.51 5.456739
#> 6 1 ICG15419 JALGOAN_R15 4.95 5.187674
unique(PredictionsSYPP$Fold)
#> [1] 1 2 3 4 5 6 7
# Summaries
summariesPYPP <- gs_summaries(PredictionsPYPP)
summariesSYPP <- gs_summaries(PredictionsSYPP)

# Elements of summaries
names(summariesPYPP)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summariesPYPP$line)
#> Line Observed Predicted Difference
#> 1 ICGV95377 9.4725 9.4803 0.0078
#> 2 CSMG84-1 8.9550 8.9401 0.0149
#> 3 TG19 9.0500 8.9692 0.0808
#> 4 ICG9315 8.6350 8.7362 0.1012
#> 5 ICGV99085 8.8075 8.9762 0.1687
#> 6 DTG3 9.5325 9.3563 0.1762

head(summariesSYPP$line)
#> Line Observed Predicted Difference
#> 1 ICG10036 4.5525 4.6444 0.0919
#> 2 ICGV99085 5.1425 5.2460 0.1035
#> 3 ICGV95377 5.7250 5.5596 0.1654
#> 4 ICG3343 5.5150 5.3147 0.2003
#> 5 CSMG84-1 4.8825 5.1152 0.2327
#> 6 ICG9315 4.9400 5.1753 0.2353

# Summaries by Environment
summariesPYPP$env[, 1:8]
#> Env MSE MSE_SE RMSE RMSE_SE NRMSE
#> 1 ALIYARNAGAR_R15 14.8938 3.4999 3.7014 0.4460 1.1295
#> 2 ICRISAT_PR15-16 8.4401 1.3024 2.8529 0.2241 1.9954
#> 3 ICRISAT_R15 11.5754 3.7984 2.9970 0.6574 1.0463

```

```

#> 4      JALGOAN_R15 23.8182 5.3977 4.6463 0.6097 1.0197
#> 5      Global 13.4244 1.8769 3.6044 0.2685 0.9294
#>      NRMSE_SE      MAE
#> 1 0.1786 3.2208
#> 2 0.5544 2.4984
#> 3 0.3144 2.4327
#> 4 0.1051 3.7597
#> 5 0.0928 2.8556
summariesSYPP$env[, 1:8]
#>      Env      MSE MSE_SE      RMSE RMSE_SE      NRMSE NRMSE_SE
#> 1 ALIYARNAGAR_R15 6.0259 1.7731 2.2897 0.3613 1.2928 0.3149
#> 2 ICRISAT_PR15-16 3.5385 0.6567 1.8248 0.1864 1.8315 0.4978
#> 3      ICRISAT_R15 4.2019 1.4021 1.8867 0.3272 0.8446 0.1121
#> 4      JALGOAN_R15 9.5201 2.2852 2.9389 0.3836 1.1481 0.1789
#> 5      Global 5.3769 0.5858 2.2972 0.1290 0.9324 0.0899
#>      MAE
#> 1 1.9142
#> 2 1.6113
#> 3 1.5278
#> 4 2.4488
#> 5 1.8132

# Summaries by Fold
summariesPYPP$fold[, 1:8]
#>      Fold      MSE MSE_SE      RMSE RMSE_SE      NRMSE NRMSE_SE      MAE
#> 1      1 7.4018 3.2823 2.3941 0.7462 0.8765 0.2671 2.0421
#> 2      2 10.2763 3.6687 3.0724 0.5280 0.9231 0.1216 2.4829
#> 3      3 23.1129 4.8205 4.7072 0.5644 1.3772 0.2699 3.8732
#> 4      4 5.4316 1.6385 2.2241 0.4021 2.3948 0.9445 1.9501
#> 5      5 23.4695 8.0862 4.6193 0.8430 1.2133 0.2837 3.7905
#> 6      6 17.5410 4.5008 4.0728 0.5637 0.8547 0.0785 3.5537
#> 7      7 15.5399 6.1230 3.7558 0.6913 1.4445 0.4821 3.1528
#> 8 Global 13.4244 1.8769 3.6044 0.2685 0.9294 0.0928 2.8556
summariesSYPP$fold[, 1:8]
#>      Fold      MSE MSE_SE      RMSE RMSE_SE      NRMSE NRMSE_SE      MAE
#> 1      1 3.1893 1.4125 1.6680 0.3684 0.9751 0.2487 1.3893
#> 2      2 4.9701 2.8800 1.9920 0.5780 0.9275 0.1248 1.6186
#> 3      3 9.3970 1.4231 3.0365 0.2428 1.8262 0.5049 2.6120
#> 4      4 2.5995 0.8096 1.5443 0.2675 1.8938 0.8148 1.2499
#> 5      5 7.9440 2.1529 2.7375 0.3873 1.0598 0.1722 2.2720
#> 6      6 5.1419 1.3482 2.1924 0.3344 0.8345 0.0807 1.9621
#> 7      7 7.5095 4.2362 2.4747 0.6796 1.4380 0.4980 2.0249
#> 8 Global 5.3769 0.5858 2.2972 0.1290 0.9324 0.0899 1.8132

```

4.6 Ejemplo con kernels con Grid search y particiones aleatorias.

With grid search and random partitions.

Este ejemplo evalúa un modelo bayesiano con cinco particiones aleatorias, con 20% los datos del conjunto d prueba y 80% del conjunto de entrenamiento dentro de cada partición

(los parámetros por defecto de la función `cv_random`), para una respuesta continua, utilizando la matriz de diseño de la variable *Env* de *PhenoToy*, la matriz *G* descrita anteriormente y la matriz de diseño de la interacción entre estos dos, como predictores. Todo esto para los tipos de Kernel: "Linear", "Polynomial", "Sigmoid", "Gaussian", "Exponential", "Arc_cosine" y "Arc_cosine_L".

En este ejemplo, el conjunto de datos utilizado es *GroundnutToy* y el objetivo es predecir la variable continua *SYPP* del data frame *PhenoToy* utilizando la matriz de diseño de la variable *PhenoToy Env*, la matriz descrita *G* anteriormente y la matriz de diseño de la interacción entre estos dos, como predictores; por lo que identificamos la variable de respuesta como *y*. Tenga en cuenta que, a diferencia de los ejemplos anteriores, la variable predictora aún no se ha identificado; porque en este ejemplo es la matriz *G* descrita anteriormente a la que se aplica cada uno de los kernels.

```
# Load the data
load("GroundnutToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Line & Env
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
y <- PhenoToy$SYPP

print(y[1:7])
#> [1] 7.23 3.97 3.74 4.03 6.70 2.07 3.97
typeof(y)
#> [1] "double"
```

Tenga en cuenta que la variable de respuesta *y* es una variable continua (un vector con elementos de tipo "doble"), lo cual es importante para que el modelo se entrene automáticamente para una variable continua.

A diferencia de los ejemplos anteriores, ahora buscamos evaluar el modelo para cada tipo de kernel mencionado anteriormente. Por ello creamos un vector en el que indicamos los tipos de kernel que queremos aplicar a la matriz *G* descrita anteriormente. Además, creamos las listas vacías *PredictionsAll*, *TimesAll* y *SummariesAll* que se utilizarán para guardar las predicciones, los tiempos de ejecución y los resúmenes de cada modelo entrenado, es decir, para cada tipo de kernel; lo que a su vez servirá para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
kernels <- c(
  "linear",
  "polynomial",
  "sigmoid",
  "Gaussian",
  "exponential",
```

```

"arc_cosine",
"Arc_cosine_L"
)

# Empty lists that will contain Predictions,
# Times of execution & Summaries for each type of kernel
PredictionsAll <- list()
TimesAll <- list()
SummariesAll <- list()

```

Posteriormente, se seguirá el siguiente proceso **para cada tipo de Kernel**:

1. identifique la *variable arc_deep* con el valor 2. Si el tpo de Kernel es “Arc_cosine_L” el valor de la variable *arc_deep* se cambia a 3 y el *kernel_type* se identifica como “Arc_cosine”; de lo contrario, *kernel_type* se identifica como el kernel predeterminado.
2. El tipo de kernel establecido en (1) se aplica a la matriz de información genómica GenoToy, asignando al argumento *arc_cosine_deep* el valor establecido en la variable *arc_deep*. Tenga en cuenta que el argumento *arc_cosine_deep* se ignora si el tipo de kernel no es *Arc_cosine*.
3. Con la matriz de información kernelizada *Geno* se calcula la matriz *G* descrita anteriormente y la matriz de diseño de la interacción entre ésta y la matriz de diseño del efecto del ambiente. Además, identificamos la lista *X* en la que se especifica que el efecto del ambiente se modelará como un efecto fijo, la matriz *G* con un modelo "Bayesian Ridge Regression" (BRR) y la matriz de diseño de la interacción entre estos dos con un modelo. Bayes LASSO.
4. Luego realizamos cinco particiones aleatorias, con 80% los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función *cv_random*.
5. Predictions y *Tiempos* que se utilizarán para guardar los valores observados y predichos en cada ambiente y posteriormente evaluar la capacidad predictiva del modelo con la función *gs_summaries*, además de guardar los tiempos de ejecución de cada modelo entrenado para cada partición.
6. **Para cada partición:**
 1. El conjunto de entrenamiento y el conjunto de prueba se identifican a través de los índices del conjunto de prueba que sirven como argumento en el entrenamiento del modelo (en el caso de la función *bayesian_model*);
 2. El modelo se entrena con el conjunto de entrenamiento, indicando con la función *bayesian_model* una lista que especifica la matriz de predictores y el modelo, además de la variable respuesta y los índices correspondientes al conjunto de prueba;

3. Con el modelo obtenido en (2), se predice la variable respuesta y en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
 4. Identificación de las predicciones del conjunto de prueba: Se crea el data frame *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba. Además, cada fila de *FoldPrediction* se agrega al data frame *Predictions*.
 5. Identificación del tiempo de ejecución del entrenamiento del modelo obtenido en (2): Se crea el data frame *FoldTime* que contiene la columna que especifica el tipo de *Kernel* utilizado para entrenar el modelo, el número de *Fold* y la columna *Minutos que indica el tiempo de ejecución*, en minutos, del entrenamiento del modelo obtenido en (2). Además, cada fila de *FoldTime* se agrega al data frame *Times*.
- data frame de predicciones contiene columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición, correspondiente al formato necesario para usar la función *gs_summaries* en estas predicciones en el caso de variables continuas.
7. *Predictions* y con la ayuda de la función *gs_summaries*, evaluamos la capacidad predictiva del modelo entrenado para el tipo de kernel especificado. La función *gs_summaries* devuelve una lista que identificamos como *summaries* y que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.
 8. Finalmente, se crea un elemento con el nombre del kernel especificado en cada una de las listas *PredictionsAll*, *TimesAll* y *SummariesAll*, que corresponden a los data frames de las listas *Predictions*, *Times* y resúmenes, respectivamente.

```
for (kernel in kernels) {
  cat("*** Kernel:", kernel, "***\n")

  # Identify the arc_deep and the kernel
  arc_deep <- 2
  if (kernel == "Arc_cosine_L") {
    arc_deep <- 3
    kernel <- "arc_cosine"
  } else {
    kernel <- kernel
  }

  # Compute the kernel of th genomic relationship matrix
  Geno <- kernelize(
    # First column is Line
    GenoToy[, -1],
    kernel = kernel,
    arc_cosine_deep = arc_deep
  )
}
```

```

gene <- cholesky(Geno)
# G matrix
LinexGeno <- Line %**% Geno
LinexGenoxEnv <- model.matrix(~ 0 + LinexGeno:Env)

# Identify the model
X <- list(
  Send = list(x = Env, model = "FIXED"),
  LinexGeno = list(x = LinexGeno, model = "BRR"),
  LinexGenoxEnv = list(x = LinexGenoxEnv, model = "Bayes_Lasso")
)

# Random Partition
set.seed(2022)
folds <- cv_random(
  records_number = length(y),
  folds_number = 5,
  testing_proportion = 0.2
)

# Empty data frames that will contain Predictions & Times
# of execution for each partition
Predictions <- data.frame()
Times <- data.frame()

for (i in seq_along(folds)) {
  cat("\t*** Fold:", i, "***\n")

  # Identify the training and testing indices
  fold <- folds[[i]]

  # Model training:
  # This function receives the whole data and the
  # testing indices separately
  model <- bayesian_model(
    x = X,
    y = y,
    testing_indices = fold$testing,
    iterations_number = 1000,
    burn_in = 500
  )

  # Extract the predicted values of testing
  predictions <- predict(model)

  FoldPredictions <- data.frame(
    Fold = i,
    Line = PhenoToy$Line[fold$testing],

```

```

    Env = PhenoToy$Env[fold$testing],
    Observed = y[fold$testing],
    Predicted = predictions$predicted
  )
  Predictions <- rbind(Predictions, FoldPredictions)

  # Execution times
  FoldTime <- data.frame(
    kernel = kernel,
    Fold = i,
    Minutes = as.numeric(model$execution_time, units = "mins")
  )
  Times <- rbind(Times, FoldTime)
}
# Summaries of the Folds
summaries <- gs_summaries(Predictions)

# Predictions, Times of execution & Summaries for the
# specified Kernel
PredictionsAll[[kernel]] <- Predictions
TimesAll[[kernel]] <- Times
SummariesAll[[kernel]] <- summaries
}
#> *** Kernel: linear ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.2205 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1956 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1786 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1932 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.2027 secs ***
#> *** Kernel: polynomial ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.2014 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.194 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1845 secs ***
#> *** Fold: 4 ***

```

```

#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1827 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1788 secs ***
#> *** Kernel: sigmoid ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1822 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1794 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1898 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1827 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1901 secs ***
#> *** Kernel: Gaussian ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.197 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1834 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1938 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1869 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1872 secs ***
#> *** Kernel: exponential ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1828 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1821 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1886 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1844 secs ***

```



```

#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.176 secs ***
#> *** Kernel: arc_cosine ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1904 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1852 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1915 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1862 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.3253 secs ***
#> *** Kernel: Arc_cosine_L ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1847 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1833 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1801 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1817 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.1821 secs ***

```

Recordando que este proceso se realizó para cada tipo de kernel, cada una de las listas *PredictionsAll*, *TimesAll* y *SummariesAll* contiene las predicciones, los tiempos de ejecución y los resúmenes, respectivamente, para cada tipo de kernel aplicado a la matriz de datos *X*. A modo de ejemplo, a continuación se muestran los resultados obtenidos para el tipo de kernel “Polinomial”:

```

# Last predictions for the Polynomial Kernel
head(PredictionsAll$polynomial)
#>   Fold   Line      Env Observed Predicted
#> 1    1 ICGV97115  JALGOAN_R15    3.50  7.446200
#> 2    1  ICG9315   ICRISAT_R15    4.33  6.095577
#> 3    1 ICGV06099 ICRISAT_PR15-16    7.73  4.265610
#> 4    1 ICGV00248   ICRISAT_R15    6.59  6.635190
#> 5    1 ICGV05057   ICRISAT_R15   11.66  5.394338

```

```

#> 6      1 ICGV02434      JALGOAN_R15      1.60  6.065194
# Times of execution for the Polynomial Kernel
TimesAll$polynomial
#>      kernel Fold      Minutes
#> 1 polynomial      1 0.003233536
#> 2 polynomial      2 0.002904693
#> 3 polynomial      3 0.002782027
#> 4 polynomial      4 0.002976485
#> 5 polynomial      5 0.003143617
# Elements of SummariesAll
names(SummariesAll)
#> [1] "linear"      "polynomial"    "sigmoid"      "Gaussian"
#> [5] "exponential"  "arc_cosine"
# Elements of summaries for the Polynomial Kernel
names(SummariesAll$polynomial)
#> [1] "line" "env"  "fold"
# Summaries by Polynomial
head(SummariesAll$polynomial$line)
#>      Line Observed Predicted Difference
#> 1      49x37-134  5.4850    5.5925    0.1075
#> 2      ICGV99085  4.8533    4.9891    0.1358
#> 3      Gangapuri  4.2633    4.5667    0.3033
#> 4      CSMG84-1   6.6700    6.2500    0.4200
#> 5      ICGV07148  6.4500    6.0187    0.4313
#> 6 49x37-99(b)taLL 3.7025    4.2165    0.5140

# Summaries by Polynomial
SummariesAll$polynomial$env[, 1:8]
#>      Env      MSE MSE_SE  RMSE RMSE_SE  NRMSE
#> 1 ALIYARNAGAR_R15  4.4398 1.2288 2.0203  0.2992 1.1681
#> 2 ICRISAT_PR15-16  3.5151 1.4112 1.7539  0.3313 1.5301
#> 3 ICRISAT_R15      4.7101 1.4100 1.9976  0.4242 0.8921
#> 4 JALGOAN_R15     11.1060 2.1524 3.2454  0.3785 1.0332
#> 5 Global           5.9014 1.1297 2.3768  0.2510 1.0120
#>      NRMSE_SE      MAE
#> 1  0.0822 1.7087
#> 2  0.5272 1.5605
#> 3  0.0829 1.5534
#> 4  0.0774 2.7899
#> 5  0.0806 1.8714
SummariesAll$polynomial$env[, 9:15]
#>      MAE_SE      Cor Cor_SE Intercept Intercept_SE  Slope Slope_SE
#> 1 0.2695 -0.2072 0.2498    4.5842      8.3646 0.1015    1.1931
#> 2 0.2916  0.1835 0.1342   -3.6244      6.1139 2.2350    1.7821
#> 3 0.3169  0.4223 0.1201   -1.9087      2.0194 1.2890    0.3255
#> 4 0.3133  0.1892 0.0802    0.4469      2.9407 0.9718    0.5256
#> 5 0.2235  0.2745 0.1362    2.6607      1.5500 0.5666    0.2741
SummariesAll$polynomial$env[, 16:19]
#>      R2  R2_SE  MAAPE  MAAPE_SE
#> 1 0.2924 0.0712 0.3041  0.0635

```

```
#> 2 0.1057 0.0375 0.3949 0.0573
#> 3 0.2360 0.1196 0.2775 0.0602
#> 4 0.0615 0.0298 0.4509 0.0716
#> 5 0.1495 0.0619 0.3217 0.0348

# Summaries by Polynomial
SummariesAll$polynomial$fold[, 1:8]
#>      Fold      MSE MSE_SE      RMSE RMSE_SE      NRMSE NRMSE_SE      MAE
#> 1      1 8.7915 0.6321 2.9592 0.1078 1.1387 0.1083 2.5295
#> 2      2 6.7072 2.3463 2.4298 0.5175 0.9908 0.0834 2.0909
#> 3      3 6.2205 2.5592 2.3494 0.4833 1.6830 0.6436 1.9748
#> 4      4 2.0467 0.6668 1.3308 0.3032 0.8398 0.0656 1.1232
#> 5      5 5.9478 3.3352 2.2024 0.6047 1.1272 0.1306 1.7973
#> 6 Global 5.9014 1.1297 2.3768 0.2510 1.0120 0.0806 1.8714
SummariesAll$polynomial$fold[, 9:15]
#>      MAE_SE      Cor Cor_SE Intercept Intercept_SE      Slope Slope_SE
#> 1 0.2090 0.0049 0.2257 4.6454 4.3637 0.3338 0.7225
#> 2 0.4562 0.3416 0.0855 -6.2907 6.7315 1.7446 0.9035
#> 3 0.3844 -0.0605 0.2413 5.7125 5.4923 0.0241 0.8610
#> 4 0.2745 0.4262 0.1425 -6.3589 7.1458 2.9622 2.0902
#> 5 0.4597 0.0226 0.2286 1.6642 4.9703 0.6821 0.9454
#> 6 0.2235 0.2745 0.1362 2.6607 1.5500 0.5666 0.2741
SummariesAll$polynomial$fold[, 16:19]
#>      R2 R2_SE      MAAPE MAAPE_SE
#> 1 0.1528 0.0977 0.4656 0.0865
#> 2 0.1386 0.0542 0.4274 0.0659
#> 3 0.1784 0.1105 0.3195 0.0406
#> 4 0.2426 0.1506 0.2287 0.0801
#> 5 0.1573 0.0339 0.3431 0.0578
#> 6 0.1495 0.0619 0.3217 0.0348
```

4.7 Ejemplo de con kernels con Grid search y particiones leatorias

Este ejemplo evalúa un modelo bayesiano con cinco particiones aleatorias, con 20% los datos del conjunto de prueba y 80% del conjunto de entrenamiento dentro de cada partición (los parámetros por defecto de la función `cv_random`), para una respuesta continua, utilizando la matriz de diseño de la variable *Env* de *PhenoToy*, la matriz *G* descrita anteriormente y la matriz de diseño de la interacción entre estos dos, como predictores. Todo esto con los llamados “Sparse Kernel Methods”, con las posibles combinaciones entre los tipos de Kernel “Sparse_Gaussian” y “Sparse_Arc_cosine” con las proporciones 0.5,0.6,0.7,0.8,0.9 y 1.

En este ejemplo, el conjunto de datos utilizado es *MaizeToy* y el objetivo es predecir la variable continua *Yield* del data frame *PhenoToy* utilizando la matriz de diseño de la variable *PhenoToy Env*, la matriz descrita *G* anteriormente y la matriz de diseño de la interacción entre estos dos, como predictores; por lo que identificamos la variable de respuesta como *y*. Tenga en cuenta que, a diferencia de los ejemplos anteriores, la variable predictora aún no se ha identificado; porque en este ejemplo es la matriz *G* descrita anteriormente a la que se aplica cada uno de los kernels.

```

# Load the dataset
load("MaizeToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Line & Env
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)

# Response Variable
y <- PhenoToy$Yield
print(y[1:7])
#> [1] 6.11 6.21 5.32 6.62 5.60 6.24 5.24
typeof(y)
#> [1] "double"

```

Tenga en cuenta que la variable de respuesta *y* es una variable continua (un vector con elementos de tipo "double"), o cual es importante para que el modelo se entrene automáticamente para una variable continua.

A diferencia de los ejemplos anteriores, ahora buscamos evaluar el modelo para cada una de las posibles combinaciones entre los tipos de Kernel "Sparse_Gaussian" y "Sparse_Arc_cosine" con las proporciones 0.5, 0.6, 0.7, 0.8, 0.9 y 1. Por eso creamos un vector llamado *kernels* en el que indicamos los tipos de kernels que queremos aplicar a los de la matriz *X* y otro vector llamado *lines_proportions*. Además, creamos las listas vacías *PredictionsAll*, *TimesAll* y *SummariesAll* que se utilizarán para guardar las predicciones, los tiempos de ejecución y los resúmenes de cada modelo entrenado, es decir, para cada combinación entre tipo de kernel y proporción de líneas utilizadas; lo que a su vez servirá para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```

kernels <- c("Sparse_Gaussian", "Sparse_Arc_cosine")
lines_proportions <- c(0.5, 0.6, 0.7, 0.8, 0.9, 1)

# Empty lists that will contain Predictions,
# Times of execution & Summaries for each type of kernel
PredictionsAll <- list()
TimesAll <- list()
SummariesAll <- list()

```

Posteriormente, se seguirá el siguiente proceso **para cada tipo de Kernel y para cada proporción de líneas**:

1. El conjunto de tipos de kernel se aplica a la matriz de información genómica *GenoToy*, asignado el valor numérico a la 2ª *argumento arc_cosine_deep* y 1ª *argumento rows_proportion* el valor de *rows_proportion* establecido en *rows_proportion*. Tenga en cuenta que el argumento *arc_cosine_deep* se ignora si el tipo de kernel no es *Arc_cosine*.

2. Con la matriz de información kernelizada *Geno* se calcula la matriz *G* descrita anteriormente y la matriz de diseño de la interacción entre ésta y la matriz de diseño del efecto del ambiente. Además, identificamos la lista *X* en la que se especifica que el efecto del ambiente se modelará como un efecto fijo, la matriz *G* con un modelo "Bayesian Ridge Regression" (BRR) y la matriz de diseño de la interacción entre estos dos con un modelo. Regresión de cresta bayesiana (BRR).
3. Luego realizamos cinco particiones aleatorias, con 80% los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función *cv_random*.
4. *Predictions*, *Times* e *Hyperparams* que se utilizarán para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función *gs_summaries*, además de guardar los tiempos de ejecución de cada modelo entrenado para cada partición.
5. **Para cada partición:**
 1. El conjunto de entrenamiento y el conjunto de prueba se identifican a través de los índices del conjunto de prueba que sirven como argumento en el entrenamiento del modelo (en el caso de la función *bayesian_model*);
 2. El modelo se entrena con el conjunto de entrenamiento, indicando con la función *bayesian_model* una lista que especifica la matriz de predictores y el modelo, además de la variable respuesta y los índices correspondientes al conjunto de prueba;
 3. Con el modelo obtenido en (2), se predice la variable respuesta y en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
 4. Identificación de las predicciones del conjunto de prueba: Se crea el data frame *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba. Además, cada fila de *FoldPrediction* se agrega al data frame *Predictions*.
 5. Identificación del tiempo de ejecución del entrenamiento del modelo obtenido en (2): Se crea el data frame *FoldTime* que contiene la columna que especifica el tipo de *Kernel* utilizado para entrenar el modelo, el número de *Fold* y la columna *Minutos que indica el* tiempo de ejecución, en minutos, del entrenamiento del modelo obtenido en (2). Además, cada fila de *FoldTime* se agrega al data frame *Times*.
- data frame de predicciones contiene columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición, correspondiente al formato necesario para usar la función *gs_summaries* en estas predicciones en el caso de variables continuas.

6. Predictions y con la ayuda de la función *gs_summaries*, evaluamos la capacidad predictiva del modelo entrenado para el tipo de kernel especificado. La función *gs_summaries* devuelve una lista que identificamos como *summaries* y que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.
7. Finalmente, se crea un elemento con el nombre del kernel especificado en cada una de las listas *PredictionsAll*, *TimesAll* y *SummariesAll*, que corresponden a los data frames de las listas *Predictions*, *Times* y resúmenes, respectivamente.

```
for (kernel in kernels) {
  cat("*** Kernel:", kernel, "***\n")

  for (line_proportion in lines_proportions) {
    cat("\t*** Line_Proportion:", line_proportion, "***\n")

    # Compute the kernel
    Geno <- kernelize(
      GenoToy[, -1],
      kernel = kernel,
      arc_cosine_deep = 2,
      rows_proportion = line_proportion
    )

    LinexGeno <- Line %**% Geno
    LinexGenoxEnv <- model.matrix(~ 0 + LinexGeno:Env)

    # Identify the model
    X <- list(
      Send = list(x = Env, model = "FIXED"),
      LinexGeno = list(x = LinexGeno, model = "BRR"),
      LinexGenoxEnv = list(x = LinexGenoxEnv, model = "BRR")
    )

    # Random Partition
    folds <- cv_random(
      records_number = length(y),
      folds_number = 5,
      testing_proportion = 0.2
    )

    # Empty data frames that will contain Predictions & Times
    # of execution for each partition
    Predictions <- data.frame()
    Times <- data.frame()

    for (i in seq_along(folds)) {
      cat("\t*** Fold:", i, "***\n")

      # Identify the training and testing indices
```

```

fold <- folds[[i]]

# Model training:
# This function receives the whole data and the
# testing indices separately
model <- bayesian_model(
  x = X,
  y = y,
  testing_indices = fold$testing,
  iterations_number = 500,
  burn_in = 250
)

# Testing Predictions
predictions <- predict(model)

FoldPredictions <- data.frame(
  Fold = i,
  Line = PhenoToy$Line[fold$testing],
  Env = PhenoToy$Env[fold$testing],
  Observed = y[fold$testing],
  Predicted = predictions$predicted
)
Predictions <- rbind(Predictions, FoldPredictions)

# Execution times
FoldTime <- data.frame(
  kernel = kernel,
  LinesProportion = line_proportion,
  Fold = i,
  Minutes = as.numeric(model$execution_time, units = "mins")
)
Times <- rbind(Times, FoldTime)
}

# Summaries of the Folds
summaries <- gs_summaries(Predictions)
str_line <- paste("Line_Proportion:", line_proportion)

# Predictions, Times of execution & Summaries for the
# specified Kernel & line_proportion
PredictionsAll[[kernel]][[str_line]] <- Predictions
TimesAll[[kernel]][[str_line]] <- Times
SummariesAll[[kernel]][[str_line]] <- summaries
}
}

#> *** Kernel: Sparse_Gaussian ***
#> *** Line_Proportion: 0.5 ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***

```

```

#> *** Model evaluation completed in 0.0539 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0511 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0536 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0431 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0438 secs ***
#> *** Line_Proportion: 0.6 ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0453 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0443 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0573 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0465 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0446 secs ***
#> *** Line_Proportion: 0.7 ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0543 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0508 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0639 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0535 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0534 secs ***
#> *** Line_Proportion: 0.8 ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.052 secs ***
#> *** Fold: 2 ***

```



```

#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0488 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0575 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0494 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0495 secs ***
#> *** Line_Proportion: 0.9 ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0495 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0507 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0619 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0525 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0494 secs ***
#> *** Line_Proportion: 1 ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0494 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0517 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0637 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0531 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0512 secs ***
#> *** Kernel: Sparse_Arc_cosine ***
#> *** Line_Proportion: 0.5 ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0445 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***

```

```

#> *** Model evaluation completed in 0.0429 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0704 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0462 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0456 secs ***
#> *** Line_Proportion: 0.6 ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0461 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0475 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0496 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0548 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0455 secs ***
#> *** Line_Proportion: 0.7 ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0483 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0465 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0591 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0514 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0486 secs ***
#> *** Line_Proportion: 0.8 ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.047 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.049 secs ***
#> *** Fold: 3 ***

```

```

#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0576 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0532 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0498 secs ***
#> *** Line_Proportion: 0.9 ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0494 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0493 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0582 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0497 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0495 secs ***
#> *** Line_Proportion: 1 ***
#> *** Fold: 1 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0507 secs ***
#> *** Fold: 2 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0512 secs ***
#> *** Fold: 3 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0611 secs ***
#> *** Fold: 4 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0531 secs ***
#> *** Fold: 5 ***
#> *** Fitting Bayesian Model model ***
#> *** Model evaluation completed in 0.0512 secs ***

```

Recordando que este proceso se realizó paracada combinación entre el tipo de kernel especificado y la proporción de *líneas*, cada una de las listas *PredictionsAll*, *TimesAll* y *SummariesAll* contiene las predicciones, los tiempos de ejecución y los resúmenes, respectivamente, para cada combinación entre el tipo de kernel y la proporción de *líneas* aplicadas a la matriz de datos X. A modo de ejemplo, a continuación se muestran los resultados obtenidos para el tipo de kernel “Sparse_Arc_cosine” y “Line_Proportion: 0.6”:

```

# Predictions for the Sparse_Arc_cosine Kernel & Line_Proportion: 0.6
head(PredictionsAll$Sparse_Arc_cosine$`Line_Proportion: 0.6`)

```

```

#> NULL
# Times of execution for the Sparse_Arc_cosine Kernel &
# Line_Proportion: 0.6
TimesAll$Sparse_Arc_cosine$`Line_Proportion: 0.6`
#>      kernel LinesProportion Fold      Minutes
#> 1 Sparse_Arc_cosine      0.6    1 0.0007675012
#> 2 Sparse_Arc_cosine      0.6    2 0.0007917802
#> 3 Sparse_Arc_cosine      0.6    3 0.0008259098
#> 4 Sparse_Arc_cosine      0.6    4 0.0009136160
#> 5 Sparse_Arc_cosine      0.6    5 0.0007586638

# Elements of SummariesAll
names(SummariesAll)
#> [1] "Sparse_Gaussian" "Sparse_Arc_cosine"
# Elements of summaries for Sparse_Arc_cosine Kernel &
# Line_Proportion: 0.6
names(SummariesAll$Sparse_Arc_cosine)
#> [1] "Line_Proportion: 0.5" "Line_Proportion: 0.6"
#> [3] "Line_Proportion: 0.7" "Line_Proportion: 0.8"
#> [5] "Line_Proportion: 0.9" "Line_Proportion: 1"
names(SummariesAll$Sparse_Arc_cosine$`Line_Proportion: 0.6`)
#> [1] "line" "env" "fold"

# Summaries by Line
head(SummariesAll$Sparse_Arc_cosine$`Line_Proportion: 0.6`$line)
#>      Line Observed Predicted Difference
#> 1 CKDHL0529    5.640    5.6198    0.0202
#> 2 CKDHL0136    5.990    5.9150    0.0750
#> 3 CKDHL0203    5.535    5.6145    0.0795
#> 4 CKDHL0027    5.834    5.7525    0.0815
#> 5 CKDHL0150    6.384    6.4673    0.0833
#> 6 CKDHL0433    6.270    6.1466    0.1234

# Summaries by Enviroment
SummariesAll$Sparse_Arc_cosine$`Line_Proportion: 0.6`$env[, 1:8]
#>      Env      MSE MSE_SE  RMSE RMSE_SE  NRMSE NRMSE_SE  MAE
#> 1  EBU 0.3797 0.1563 0.5725 0.1139 0.8679 0.0774 0.4619
#> 2  KAK 0.7113 0.2580 0.7919 0.1451 1.0109 0.0278 0.6414
#> 3  KTI 2.0402 0.3752 1.4027 0.1349 0.9934 0.0410 1.2670
#> 4 Global 0.9243 0.1090 0.9545 0.0572 0.8945 0.0394 0.7052
SummariesAll$Sparse_Arc_cosine$`Line_Proportion: 0.6`$env[, 9:15]
#>      MAE_SE      Cor Cor_SE Intercept Intercept_SE  Slope
#> 1 0.1062 0.5524 0.1269 -17.8226 16.3218 3.7808
#> 2 0.1274 -0.3406 0.1950 20.0537 10.6387 -2.8938
#> 3 0.1462 0.0966 0.2417 -16.7039 26.8994 3.7579
#> 4 0.0517 0.4394 0.1015 0.1257 1.4228 0.9870
#>      Slope_SE
#> 1 2.5433
#> 2 2.0633
#> 3 4.5069

```

```

#> 4 0.2403
SummariesAll$Sparse_Arc_cosine$`Line_Proprtion: 0.6`$env[, 16:19]
#>      R2  R2_SE  MAAPE  MAAPE_SE
#> 1 0.3696 0.1312 0.0729 0.0171
#> 2 0.2681 0.1914 0.1343 0.0266
#> 3 0.2430 0.0826 0.2264 0.0333
#> 4 0.2343 0.0879 0.1259 0.0035

# Summaries by Fold
SummariesAll$Sparse_Arc_cosine$`Line_Proprtion: 0.6`$fold[, 1:8]
#>      Fold  MSE MSE_SE  RMSE RMSE_SE  NRMSE NRMSE_SE  MAE
#> 1      1 1.0665 0.2947 1.0128 0.1428 0.9962 0.0485 0.8809
#> 2      2 0.5759 0.2200 0.7286 0.1500 0.9442 0.1130 0.5661
#> 3      3 0.9564 0.7658 0.8158 0.3814 0.9541 0.0438 0.7108
#> 4      4 0.8736 0.5516 0.8245 0.3114 0.8727 0.1099 0.6998
#> 5      5 1.7463 0.8118 1.2301 0.3413 1.0199 0.0529 1.0930
#> 6 Global 0.9243 0.1090 0.9545 0.0572 0.8945 0.0394 0.7052
SummariesAll$Sparse_Arc_cosine$`Line_Proprtion: 0.6`$fold[, 9:15]
#>      MAE_SE      Cor Cor_SE Intercept Intercept_SE  Slope
#> 1 0.1002 -0.2193 0.4809 -15.8178 33.2719 3.1870
#> 2 0.1656 0.4000 0.3160 -4.8917 8.6927 1.5894
#> 3 0.3416 0.1944 0.1221 -0.0602 3.4086 0.9940
#> 4 0.3139 0.4687 0.2120 -38.7967 39.1232 7.4424
#> 5 0.3545 -0.3297 0.2363 35.4452 16.3364 -5.4713
#> 6 0.0517 0.4394 0.1015 0.1257 1.4228 0.9870
#>      Slope_SE
#> 1 5.3330
#> 2 1.5027
#> 3 0.6114
#> 4 6.5757
#> 5 3.2107
#> 6 0.2403
SummariesAll$Sparse_Arc_cosine$`Line_Proprtion: 0.6`$fold[, 16:19]
#>      R2  R2_SE  MAAPE  MAAPE_SE
#> 1 0.5106 0.2629 0.1546 0.0104
#> 2 0.3597 0.2232 0.1111 0.0336
#> 3 0.0676 0.0528 0.1175 0.0565
#> 4 0.3096 0.1546 0.1232 0.0472
#> 5 0.2204 0.1006 0.2162 0.0809
#> 6 0.2343 0.0879 0.1259 0.0035

```

5 Random forest

5.1 Ejemplo con respuesta continua con Grid search y particiones aleatorias solo G en el predictor

Este ejemplo evalúa un modelo Random Forest con cinco particiones aleatorias, con 20% los datos para el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de

cada partición (los parámetros predeterminados de la función `cv_random`), para una respuesta continua, usando solo la matriz *G* (Matriz de diseño de línea que contiene información genómica) como predictor y usando "Grid Search" como tipo de tuneo para los hiperparámetros *tree_number* y *node_size*.

En este ejemplo, el conjunto de datos utilizado es *ChickpeaToy* y el objetivo es predecir la variable continua *AvePlantHeight* del marco de datos *PhenoToy* utilizando la matriz *G* descrita anteriormente como predictor; por lo que identificamos las variables predictoras y de respuesta como *X* y *y* respectivamente.

```
# Load the data
load("ChickpeaToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy
# Data preparation of G
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %%% Geno

# Predictor and Response Variables
X <- LineG
y <- PhenoToy$AvePlantHeight

# Note that y is a continuous numeric vector
class(y)
#> [1] "numeric"
typeof(y)
#> [1] "double"
```

Tenga en cuenta que la variable de respuesta *y* es una variable continua (un vector con elementos de tipo "doble"), lo cual es importante para que el modelo se entrene automáticamente para este tipo de variable.

Posteriorment, realizamos cinco particiones aleatorias, con 80% los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función `cv_random` (con los parámetros por defecto). Además, creamos los data frames vacíos *Predictions* e *Hyperparams* que se utilizarán para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
# Set seed for reproducible results
set.seed(2022)
folds <- cv_random(records_number = nrow(X))

# A data frame that will contain the variables:
Predictions <- data.frame()
Hyperparams <- data.frame()
```

Posteriormente, se seguirá el siguiente proceso **paracada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo los valores 30, 50 y 80 para el hiperparámetro *tree_number* y los valores 50, 100 y 150 para el hiperparámetro *node_size*, con "Grid Search" como tipo de tuneo (parámetro predeterminado de *tune_type*). Cabe señalar que estos no son los únicos hiperparámetros ajustables en el modelo;
3. Con el modelo obtenido en (2), se predice la variable respuesta *AvePlantHeight* en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba.
 - b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*.
5. Identificación de hiperparámetros:
 - a. *HyperparamsFold* que contiene las columnas *tree_number*, *node_size*, *mse* y *Fold*, donde *mse* corresponde al costo del modelo para cada combinación de los hiperparámetros especificados y el número de *Fold*.
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
6. Se muestran los hiperparámetros óptimos del modelo obtenido en (2).

```
# Model training and predictions of the ith partition
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")
  fold <- folds[[i]]

  # Identify the training and testing sets
  X_training <- X[fold$training, ]
  X_testing <- X[fold$testing, ]
  y_training <- y[fold$training]
  y_testing <- y[fold$testing]

  # Model training
  model <- random_forest(
    x = X_training,
    y = y_training,

    # Specify the hyperparameters
    trees_number = c(30, 50, 80),
```



```

    node_size = c(50, 100, 150),
    tune_type = "grid_search"
  )

  # Prediction of the test set
  predictions <- predict(model, X_testing)

  # Predictions for the Fold Fold
  FoldPredictions <- data.frame(
    Fold = i,
    Line = PhenoToy$Line[fold$testing],
    Env = PhenoToy$Env[fold$testing],
    Observed = y_testing,
    Predicted = predictions$predicted
  )
  Predictions <- rbind(Predictions, FoldPredictions)

  # Hyperparams
  HyperparamsFold <- model$hyperparams_grid %>%
    mutate(Fold = i)
  Hyperparams <- rbind(Hyperparams, HyperparamsFold)

  # Best hyperparams of the model
  cat("*** Optimal hyperparameters: ***\n")
  print(model$best_hyperparams)
}

#> *** Fold: 1 ***
#> *** Grid Search Tuning ***
#> Total combinations: 9
#> Combination: 1 / 9
#>   KFoldCV: 1 / 5
#>   KFoldCV: 2 / 5
#>   KFoldCV: 3 / 5
#>   KFoldCV: 4 / 5
#>   KFoldCV: 5 / 5
#> Combination: 2 / 9
#>   KFoldCV: 1 / 5
#>   KFoldCV: 2 / 5
#>   KFoldCV: 3 / 5
#>   KFoldCV: 4 / 5
#>   KFoldCV: 5 / 5
#> Combination: 3 / 9
#>   KFoldCV: 1 / 5
#>   KFoldCV: 2 / 5
#>   KFoldCV: 3 / 5
#>   KFoldCV: 4 / 5
#>   KFoldCV: 5 / 5
#> Combination: 4 / 9
#>   KFoldCV: 1 / 5

```



```

#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 5 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 6 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 7 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 8 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 9 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> *** Fitting Random Forest model ***
#> *** Model evaluation completed in 0.6347 secs ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 30
#>
#> $node_size
#> [1] 50
#>
#> $mse
#> [1] 80.99386
#>
#> *** Fold: 2 ***
#> *** Grid Search Tuning ***
#> Total combinations: 9
#> Combination: 1 / 9

```

```

#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 2 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 3 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 4 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 5 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 6 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 7 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 8 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 9 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5

```

```

#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> *** Fitting Random Forest model ***
#> *** Model evaluation completed in 0.6237 secs ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 30
#>
#> $node_size
#> [1] 150
#>
#> $mse
#> [1] 80.62419
#>
#> *** Fold: 3 ***
#> *** Grid Search Tuning ***
#> Total combinations: 9
#> Combination: 1 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 2 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 3 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 4 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 5 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 6 / 9
#>      KFoldCV: 1 / 5

```

```

#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 7 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 8 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 9 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> *** Fitting Random Forest model ***
#> *** Model evaluation completed in 0.5967 secs ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 30
#>
#> $node_size
#> [1] 150
#>
#> $mse
#> [1] 87.86922
#>
#> *** Fold: 4 ***
#> *** Grid Search Tuning ***
#> Total combinations: 9
#> Combination: 1 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 2 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 3 / 9

```

```

#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 4 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 5 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 6 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 7 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 8 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 9 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> *** Fitting Random Forest model ***
#> *** Model evaluation completed in 0.5627 secs ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 80
#>
#> $node_size
#> [1] 150
#>

```

```

#> $mse
#> [1] 90.22789
#>
#> *** Fold: 5 ***
#> *** Grid Search Tuning ***
#> Total combinations: 9
#> Combination: 1 / 9
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 2 / 9
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 3 / 9
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 4 / 9
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 5 / 9
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 6 / 9
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 7 / 9
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 8 / 9
#>     KFoldCV: 1 / 5

```

```

#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 9 / 9
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> *** Fitting Random Forest model ***
#> *** Model evaluation completed in 0.6056 secs ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 30
#>
#> $node_size
#> [1] 50
#>
#> $mse
#> [1] 81.70019

```

Predictions contiene las columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo los hiperparámetros óptimos (entre los valores posibles de las combinaciones de *tree_numbe* y *node_size*) que minimizar la función de costo con el tipo de tuneo "Grid Search", correspondiente al formato necesario para usar la función *gs_summaries* en estas predicciones en el caso de variables continuas.

La función *gs_summaries* devuelve una lista que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```

head(Predictions)
#>      Fold      Line Env Observed Predicted
#> 1      1 ICCV97301   6 65.33333   48.5825
#> 2      1 ICCV04103   1 47.56667   48.5825
#> 3      1 ICCV05109   4 52.66667   48.5825
#> 4      1 ICCV00402   7 38.00000   48.5825
#> 5      1 ICCV09114   4 51.33333   48.5825
#> 6      1 ICCV03102   2 52.30000   48.5825
unique(Predictions$Fold)
#> [1] 1 2 3 4 5
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env"  "fold"

# Summaries by Line
head(summaries$line)

```

```

#>      Line Observed Predicted Difference
#> 1 ICCV05109  48.6333   48.8707    0.2373
#> 2 ICCV04312  48.4417   48.7080    0.2663
#> 3 ICCV03309  48.6000   48.9504    0.3504
#> 4 ICCV01301  49.1000   48.7071    0.3929
#> 5 ICCV05307  48.4250   48.8407    0.4157
#> 6 ICCV09114  49.6852   48.8032    0.8820

# Summaries by Environment
summaries$env[, 1:7]
#>      Env      MSE  MSE_SE    RMSE RMSE_SE  NRMSE  NRMSE_SE
#> 1      1  95.3828  6.5181   9.7441  0.3300  1.8674  0.1728
#> 2      2 171.3175 31.6824 12.7952  1.3783  2.1820  0.1812
#> 3      4  50.2729 18.0094   6.5996  1.2960  1.8150  0.2115
#> 4      5  57.2660 11.6336   7.3538  0.8927  1.9813  0.2897
#> 5      6 100.4218 18.2079   9.8627  0.8873  1.7839  0.3077
#> 6      7  56.4522 13.8648   7.2566  0.9740  2.1277  0.1810
#> 7 Global  70.3468  5.8872   8.3586  0.3469  1.0481  0.0204
summaries$env[, 8:14]
#>      MAE MAE_SE Cor Cor_SE Intercept Intercept_SE Slope
#> 1  8.5655 0.4614 NaN    NA   40.4405      0.6148  NaN
#> 2 11.5431 1.2503 NaN    NA   60.2615      1.0989  NaN
#> 3  5.6018 1.1942 NaN    NA   54.2802      1.1955  NaN
#> 4  6.6985 0.8027 NaN    NA   42.2785      0.6624  NaN
#> 5  8.9955 0.8840 NaN    NA   56.4200      1.4114  NaN
#> 6  6.6297 0.9366 NaN    NA   42.1543      0.8273  NaN
#> 7  6.7332 0.4564 NaN    NA   49.0096      1.2589  NaN
summaries$env[, 15:19]
#>      Slope_SE R2 R2_SE  MAAPE MAAPE_SE
#> 1      NA NaN    NA 0.2214  0.0125
#> 2      NA NaN    NA 0.1801  0.0164
#> 3      NA NaN    NA 0.0970  0.0183
#> 4      NA NaN    NA 0.1636  0.0215
#> 5      NA NaN    NA 0.1527  0.0109
#> 6      NA NaN    NA 0.1623  0.0252
#> 7      NA NaN    NA 0.1397  0.0074

# Summaries by Fold
summaries$fold[, 1:8]
#>      Fold      MSE  MSE_SE    RMSE RMSE_SE  NRMSE  NRMSE_SE
#> 1      1 102.1756 33.9016   9.3590  1.7079  2.1319  0.1791
#> 2      2  82.2621 22.6167   8.5962  1.2936  2.1275  0.0915
#> 3      3  95.4601 30.8785   9.0873  1.6051  1.8565  0.1941
#> 4      4  63.1671 11.4299   7.7423  0.8029  1.8662  0.2630
#> 5      5  99.5292 12.3553   9.8919  0.5797  1.8400  0.2421
#> 6 Global  70.3468  5.8872   8.3586  0.3469  1.0481  0.0204
#>      MAE
#> 1 8.3975
#> 2 7.8106
#> 3 8.1572

```



```
#> 4 6.8360
#> 5 8.8271
#> 6 6.7332
```

Además, Hyperparams contiene las columnas *tree_number*, *node_size*, *mse* y *Fold*, donde el valor de la columna *mse* corresponde al costo del modelo para cada combinación de hiperparámetros y partición, ordenados de menor a mayor dentro de cada partición.

```
# First rows of Hyperparams
head(Hyperparams)
#>   trees_number node_size      mse Fold
#> 1           30        50 80.99386    1
#> 2           50        50 81.13108    1
#> 4           30       100 81.17245    1
#> 7           30       150 81.20453    1
#> 9           80       150 81.30129    1
#> 5           50       100 81.31647    1

# Last rows of Hyperparams
tail(Hyperparams)
#>   trees_number node_size      mse Fold
#> 44           30       100 81.80703    5
#> 64           80       100 81.81778    5
#> 94           80       150 81.88470    5
#> 84           50       150 81.92386    5
#> 74           30       150 82.08621    5
#> 54           50       100 82.09748    5
```

5.2 Ejemplo con respuesta binaria con Grid search y validación cruzada 7-fold con *Env* + *G* el predictor

Este ejemplo evalúa un modelo de Random Forest con validación cruzada de 7 veces, para una respuesta binaria, utilizando el efecto Environment y la matriz *G* como predictores, así como "Grid Search" como tipo de ajuste para los hiperparámetros *tree_number* y *node_size*.

En este ejemplo, el conjunto de datos utilizado es *MaizeToy* y el objetivo es predecir la variable binaria y_{bin} , que es una transformación de la variable *Yield* * de *PhenoToy*, indicando si la respuesta es mayor que la mediana de esta variable o no, usando un diseño matriz de la variable *PhenoToy Env* y la matriz, *G* descrita anteriormente, como predictores; por lo que identificamos las variables predictoras y de respuesta como *X* y y_{bin} respectivamente.

```
# Load the data
load("MaizeToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env &
```

```

Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %*% Geno

# Predictor and Response Variables
# Predictor and Response Variables
X <- cbind(Env, LineG)
y_bin <- BurStMisc::ntile(PhenoToy$Yield, 2, result = "factor")

```

Tenga en cuenta que la variable de respuesta y_{bin} es un factor con solo dos niveles (o categorías), lo cual es importante para que el modelo se entrene automáticamente para una variable binaria (regresión logística). Por esta razón, es importante tener en cuenta esas variables de respuesta binarias o categóricas antes de usar la función *random_forest*.

Posteriormente hacemos las particiones correspondientes a 7-Fold CV, con la ayuda de la función *cv_kfold*. Además, creamos los data frames vacíos Predictions e Hyperparams que se utilizarán para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```

# Set seed for reproducible results
set.seed(2022)
folds <- cv_kfold(records_number = nrow(X), k = 7)

# A data frame that will contain the variables:
Predictions <- data.frame()
Hyperparams <- data.frame()

```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo los valores 50, 7, 100 y 150 para el hiperparámetro *tree_number* y los valores 5, 10, 15 y 20 para el hiperparámetro *node_size*, con "Grid Search" como tipo de tuneo (parámetro predeterminado de *tune_type*). Cabe señalar que estos no son los únicos hiperparámetros ajustables en el modelo;
3. Con el modelo obtenido en (2), y_{bin} se predice la variable respuesta en el conjunto de prueba, con el fin de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: *Fold number*, *Line*, *Env*, *Observed*, *Predicted*, 1 y 2 para cada elemento del conjunto de prueba. Tenga en cuenta que, a diferencia del ejemplo anterior, ahora tenemos dos columnas adicionales

correspondientes a las probabilidades asociadas con cada elemento correspondiente a esa categoría.

- b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*. Con esto, *Predictions* se formatea para que sea un argumento de la función *gs_summaries*.
5. Identificación de hiperparámetros:
 - a. *HyperparamsFold* que contiene las columnas *tree_number*, *node_size*, *accuracy* y *Fold*, donde *accuracy* es la precisión del modelo para cada combinación de los hiperparámetros especificados y el número de *Fold*.
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
6. Se muestran los hiperparámetros óptimos.

```
# Model training and predictions of the ith partition
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")
  fold <- folds[[i]]

  # Identify the training and testing sets
  X_training <- X[fold$training, ]
  X_testing <- X[fold$testing, ]
  y_training <- y_bin[fold$training]
  y_testing <- y_bin[fold$testing]

  # Model training
  model <- random_forest(
    x = X_training,
    y = y_training,

    # Specify the hyperparameters
    trees_number = c(50, 70, 100, 150),
    node_size = c(5, 10, 15, 20),
    tune_type = "grid_search",
    # In this example the iterations wont be shown
    verbose = FALSE
  )

  # Prediction of the test set
  predictions <- predict(model, X_testing)

  # Predictions for the Fold Fold
  FoldPredictions <- cbind(
    data.frame(
      Fold = i,
      Line = PhenoToy$Line[fold$testing],
      Env = PhenoToy$Env[fold$testing],
```

```

    Observed = y_testing,
    Predicted = predictions$predicted
  ),
  predictions$probabilities
)

Predictions <- rbind(Predictions, FoldPredictions)

# Hyperparams
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)

# Best hyperparams of the model
cat("*** Optimal hyperparameters: ***\n")
print(model$best_hyperparams)
}
#> *** Fold: 1 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 50
#>
#> $node_size
#> [1] 15
#>
#> $accuracy
#> [1] 0.7775
#>
#> *** Fold: 2 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 100
#>
#> $node_size
#> [1] 15
#>
#> $accuracy
#> [1] 0.7408333
#>
#> *** Fold: 3 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 50
#>
#> $node_size
#> [1] 15
#>
#> $accuracy
#> [1] 0.6908333
#>

```

```

#> *** Fold: 4 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 100
#>
#> $node_size
#> [1] 15
#>
#> $accuracy
#> [1] 0.7566667
#>
#> *** Fold: 5 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 150
#>
#> $node_size
#> [1] 15
#>
#> $accuracy
#> [1] 0.7258333
#>
#> *** Fold: 6 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 50
#>
#> $node_size
#> [1] 15
#>
#> $accuracy
#> [1] 0.7283333
#>
#> *** Fold: 7 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 70
#>
#> $node_size
#> [1] 5
#>
#> $accuracy
#> [1] 0.7

```

Predictions contiene las columnas *Fold*, *Line*, *Env*, *Observed*, *Predicted*, 1 y 2 para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo los hiperparámetros óptimos (entre los valores posibles de las combinaciones de *tree_number* y *node_size*) que minimizan la función de costo con el tipo de tuneo Grid

Search", correspondiente al fomato necesario para usar la función *gs_summaries* en estas predicciones en el caso de variables binarias.

La función *gs_summaries* devuelve una lista que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```
head(Predictions)
#>   Fold      Line Env Observed Predicted      1      2
#> 1     1 CKDHL0032 KTI          2         2 0.4421513 0.5578487
#> 2     1 CKDHL0049 EBU          1         2 0.2563557 0.7436443
#> 3     1 CKDHL0050 EBU          1         2 0.2306162 0.7693838
#> 4     1 CKDHL0052 KTI          2         2 0.4133860 0.5866140
#> 5     1 CKDHL0085 KTI          2         2 0.4762442 0.5237558
#> 6     1 CKDHL0097 KTI          1         1 0.5192509 0.4807491
unique(Predictions$Fold)
#> [1] 1 2 3 4 5 6 7
# Summaries
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted      X1      X2
#> 1 CKDHL0027          2         2 0.4931 0.5069
#> 2 CKDHL0032          2         2 0.4706 0.5294
#> 3 CKDHL0046          1         1 0.4855 0.5145
#> 4 CKDHL0049          1         1 0.4927 0.5073
#> 5 CKDHL0050          2         1 0.5017 0.4983
#> 6 CKDHL0052          2         2 0.4021 0.5979

# Summaries by Environment
summaries$env
#>      Env      PCCC PCCC_SE      Kappa Kappa_SE BrierScore
#> 1     EBU 0.8782 0.0628 0.0000 0.0000 0.3194
#> 2     KAK 0.8893 0.0715 -0.1000 0.0535 0.2951
#> 3     KTI 0.5905 0.1183 0.3541 0.1319 0.4958
#> 4 Global 0.7372 0.0471 0.4571 0.0961 0.4309
#>      BrierScore_SE
#> 1      0.0565
#> 2      0.0541
#> 3      0.0244
#> 4      0.0247

# Summaries by Fold
summaries$fold
#>      Fold      PCCC PCCC_SE      Kappa Kappa_SE BrierScore
#> 1      1 0.8381 0.0846 0.2727 0.2227 0.3596
#> 2      2 0.7000 0.1528 0.0000 0.0000 0.4037
```

```

#> 3      3 0.8889 0.1111 0.4000      NA      0.4116
#> 4      4 0.9444 0.0556 0.5000 0.4082 0.2954
#> 5      5 0.7000 0.1528 0.1000 0.0816 0.4123
#> 6      6 0.8889 0.1111 0.3333      NA      0.2897
#> 7      7 0.5417 0.2917 -0.1000 0.0816 0.4184
#> 8 Global 0.7372 0.0471 0.4571 0.0961 0.4309
#>      BrierScore_SE
#> 1      0.1160
#> 2      0.0598
#> 3      0.0403
#> 4      0.0670
#> 5      0.0925
#> 6      0.1017
#> 7      0.1529
#> 8      0.0247

```

Además, Hyperparams contiene las columnas *tree_number*, *node_size*, *accuracy* y *Fold*, donde el valor de la columna de *precisión* corresponde al costo del modelo para cada combinación de valores de partición e hiperparámetro, ordenados de menor a mayor dentro de cada partición.

```

# First rows of Hyperparams
head(Hyperparams)
#>      trees_number node_size accuracy Fold
#> 9              50         15 0.7775000    1
#> 7              100         10 0.7775000    1
#> 12             150         15 0.7641667    1
#> 11             100         15 0.7516667    1
#> 5              50         10 0.7391667    1
#> 6              70         10 0.7383333    1

# Last rows of Hyperparams
tail(Hyperparams)
#>      trees_number node_size accuracy Fold
#> 66              70         10 0.6091667    7
#> 106             70         15 0.6091667    7
#> 156             100         20 0.4516667    7
#> 136             50         20 0.4516667    7
#> 166             150         20 0.4016667    7
#> 146             70         20 0.4016667    7

```

5.3 Ejemplo con variable respuesta categórica con optimización bayesiana con particiones aleatorias *Env* + *G* + *GE* dentro del predictor

Este ejemplo evalúa un modelo Random Forest con cinco particiones aleatorias, con 20% los datos para el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de cada partición (los parámetros predeterminados de la función `cv_random`), para una respuesta categórica, utilizando el Efecto del ambiente, la matriz *G* y la interacción entre estos

dos como predictores, además de utilizar la "Optimización Bayesiana" como un tipo de tuneo para los hiperparámetros.

En este ejemplo, el conjunto de datos utilizado es *EYTToy* y buscamos predecir la variable categórica *y*, que es una transformación de la variable *GY* del marco de datos *PhenoToy* usando la función *ntile*, usando la matriz de diseño de la variable *PhenoToy Env*, la matriz *G* descrita anterior y la matriz de diseño de la interacción entre estos dos, como predictores; por lo que identificamos las variables predictoras y de respuesta como *X* y *y* respectivamente.

```
# Load the data
load("EYTTToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env, G & GE
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
LineG <- Line %**% Geno
LinexGenoxEnv <- model.matrix(~ 0 + LineG:Env)

# Predictor and Response Variables
X <- cbind(Env, LineG, LinexGenoxEnv)
y <- BurStMisc::ntile(PhenoToy$GY, 3, result = "factor")

# First 30 responses
print(y[1:30])
#> [1] 1 2 3 1 2 2 2 1 3 2 2 1 2 2 3 1 3 3 3 1 3 3 3 1 2 3 3 1 3
#> [30] 3
#> Levels: 1 < 2 < 3
```

Tenga en cuenta que la variable de respuesta *y* es un factor con tres niveles (o categorías), lo cual es importante para que el modelo se entrene automáticamente para una variable categórica (**modelo multinomial simétrico**). Por esta razón, es importante tener en cuenta esas variables de respuesta binarias o categóricas antes de usar la función *random_forest*.

Posteriormente, realizamos cinco particiones aleatorias, con 80% los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función *cv_random* (con los parámetros por defecto). Además, creamos los data frames vacíos *Predictions* e *Hyperparams* que servirán para guardar los valores observados y predichos en cada ambiente y posteriormente evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
# Set seed for reproducible results
set.seed(2022)
folds <- cv_random(records_number = nrow(X))
```



```
# A data frame that will contain the variables:
```

```
Predictions <- data.frame()
```

```
Hyperparams <- data.frame()
```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo valores entre 5 y 30 para el hiperparámetro *tree_number* y valores 5 y 5 para el hiperparámetro *node_size*, con "Optimización bayesiana" como tipo de tuneo. Cabe señalar que estos no son los únicos hiperparámetros ajustables en el modelo;
3. Con el modelo obtenido en (2), se predice la variable respuesta y en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed*, *Predicted*, 1, 2 y 3 para cada elemento del conjunto de prueba. Tenga en cuenta que, a diferencia de los ejemplos anteriores, ahora tenemos tres columnas adicionales correspondientes a las probabilidades asociadas con cada elemento correspondiente a esa categoría.
 - b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*. Con esto, *Predictions* se formatea para que sea un argumento de la función *gs_summaries*.
5. Identificación de hiperparámetros:
 - a. *HyperparamsFold* que contiene las columnas *tree_number*, *node_size*, *accuracy* y *Fold*, donde *accuracy* es la precisión del modelo para cada combinación de los hiperparámetros especificados y el número de pliegue.
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
6. Se muestran los hiperparámetros óptimos.

```
# Model training and predictions of the ith partition
```

```
for (i in seq_along(folds)) {  
  cat("*** Fold:", i, "***\n")  
  fold <- folds[[i]]
```

```
  # Identify the training and testing sets
```

```
  X_training <- X[fold$training, ]
```

```
  X_testing <- X[fold$testing, ]
```

```
  y_training <- y[fold$training]
```

```

y_testing <- y[fold$testing]

# Model training
model <- random_forest(
  x = X_training,
  y = y_training,

  # Specify the hyperparameter ranges
  trees_number = list(min = 5, max = 30),
  node_size = list(min = 5, max = 15),
  tune_type = "Bayesian_optimization",
  tune_bayes_samples_number = 5,
  tune_bayes_iterations_number = 5,

  # In this example the iterations wont bw shown
  verbose = FALSE
)

# Testing Predictions
predictions <- predict(model, X_testing)

# Predictions for the Fold Fold
FoldPredictions <- cbind(
  data.frame(
    Fold = i,
    Line = PhenoToy$Line[fold$testing],
    Env = PhenoToy$Env[fold$testing],
    Observed = y_testing,
    Predicted = predictions$predicted
  ),
  predictions$probabilities
)
Predictions <- rbind(Predictions, FoldPredictions)

# Hyperparams
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)

# Best hyperparams of the model
cat("*** Optimal hyperparameters: ***\n")
print(model$best_hyperparams)
}
#> *** Fold: 1 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 7
#>
#> $node_size

```

```
#> [1] 11
#>
#> $accuracy
#> [1] 0.6563158
#>
#> *** Fold: 2 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 29
#>
#> $node_size
#> [1] 8
#>
#> $accuracy
#> [1] 0.6978947
#>
#> *** Fold: 3 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 22
#>
#> $node_size
#> [1] 10
#>
#> $accuracy
#> [1] 0.74
#>
#> *** Fold: 4 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 15
#>
#> $node_size
#> [1] 12
#>
#> $accuracy
#> [1] 0.6257895
#>
#> *** Fold: 5 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 30
#>
#> $node_size
#> [1] 12
#>
#> $accuracy
#> [1] 0.6684211
```

Predictions contiene las columnas *Fold*, *Line*, *Env*, *Observed*, *Predicted*, 1, 2* y 3 para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo los hiperparámetros óptimos (entre los valores posibles de las combinaciones de *trees_number* y *node_size*) que minimizan la función de costo (*pcic*: Proporción de Casos Incorrectamente Clasificados) con el tipo de tuneo "Optimización Bayesiana", correspondiente al formato necesario para usar la función *gs_summaries* en *Predicción* en l caso de variables categóricas.

La función *gs_summaries* devuelve una lista que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```
head(Predictions)
#>   Fold      Line      Env Observed Predicted      1
#> 1     1 GID7632666 FlatDrip      1         1 1.00000000
#> 2     1 GID7628158 Flat5IR      2         3 0.09255281
#> 3     1 GID7631195      EHT      3         2 0.09500379
#> 4     1 GID7628467 Flat5IR      3         3 0.10704282
#> 5     1 GID7630553 Flat5IR      3         2 0.05102041
#> 6     1 GID7629600 FlatDrip      1         1 1.00000000
#>           2           3
#> 1 0.00000000 0.00000000
#> 2 0.4425280 0.4649192
#> 3 0.5533608 0.3516354
#> 4 0.2676137 0.6253435
#> 5 0.5416667 0.4073129
#> 6 0.0000000 0.0000000
unique(Predictions$Fold)
#> [1] 1 2 3 4 5
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"

# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted      X1      X2      X3
#> 1 GID7462121      1         2 0.1043 0.5686 0.3271
#> 2 GID7625106      2         2 0.1018 0.5438 0.3544
#> 3 GID7625276      1         1 0.4113 0.3738 0.2149
#> 4 GID7625985      1         1 0.4059 0.3595 0.2346
#> 5 GID7626366      1         1 0.4988 0.1548 0.3464
#> 6 GID7626381      3         3 0.3174 0.1599 0.5227

# Summaries by Environment
summaries$env
#>      Env  PCCC PCCC_SE  Kappa Kappa_SE BrierScore
#> 1 Bed5IR 0.5000 0.0573 0.1656 0.0603 0.6307
#> 2 EHT 0.5733 0.0323 0.1824 0.0854 0.5924
```

```

#> 3 Flat5IR 0.5324 0.0799 0.0460 0.1298 0.5431
#> 4 FlatDrip 1.0000 0.0000 NaN NA 0.0615
#> 5 Global 0.6469 0.0312 0.4543 0.0361 0.4919
#> BrierScore_SE
#> 1 0.0811
#> 2 0.0183
#> 3 0.0408
#> 4 0.0060
#> 5 0.0252

# Summaries by Fold
summaries$fold
#> Fold PCCC PCCC_SE Kappa Kappa_SE BrierScore
#> 1 1 0.6488 0.1272 0.0222 0.0192 0.4407
#> 2 2 0.6250 0.1423 0.0110 0.0828 0.4267
#> 3 3 0.5905 0.1472 0.1063 0.1201 0.5479
#> 4 4 0.7250 0.1109 0.4024 0.0840 0.4068
#> 5 5 0.6679 0.1127 0.1146 0.0992 0.4627
#> 6 Global 0.6469 0.0312 0.4543 0.0361 0.4919
#> BrierScore_SE
#> 1 0.1375
#> 2 0.1352
#> 3 0.1699
#> 4 0.1223
#> 5 0.1334
#> 6 0.0252

```

Además, Hyperparams contiene columnas *tree_number*, *node_size*, *accuracy* y *Fold*, donde el valor de la columna de *precisión* corresponde a la precisión del modelo para cada combinación de valores de partición e hiperparámetro, ordenados de menor a mayor dentro de cada partición.

```

# First rows of Hyperparams
head(Hyperparams)
#> trees_number node_size accuracy Fold
#> 5 7 11 0.6563158 1
#> 6 13 11 0.6257895 1
#> 9 7 12 0.6252632 1
#> 1 22 7 0.6152632 1
#> 2 16 10 0.6042105 1
#> 3 26 7 0.5957895 1

# Last rows of Hyperparams
tail(Hyperparams)
#> trees_number node_size accuracy Fold
#> 34 17 12 0.6263158 5
#> 44 24 7 0.6252632 5
#> 94 30 11 0.6052632 5
#> 24 19 6 0.5942105 5

```

```
#> 54          10          9 0.5842105    5
#> 104         23         15 0.5842105    5
```

5.4 Ejemplo multivariado con respuestas continuas con optimización bayesiana con línea de partición aleatoria $Env + G + GE$ dentro del predictor

Este ejemplo evalúa un modelo Random Forest con cinco particiones aleatorias del conjunto de líneas, con 20% las líneas para el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de cada partición, para dos respuestas continuas, utilizando el efecto ambiente, la matriz G y la interacción entre estos dos como predictores, además de utilizar la "Optimización Bayesiana" como un tipo de tuneo de hiperparámetros.

En este ejemplo, el conjunto de datos utilizado es *GroundnutToy* y el objetivo es predecir las variables continuas *PYPP* e *YPH* del marco de datos *PhenoToy* utilizando la matriz de diseño de la variable *PhenoToy Env*, la matriz G y la matriz de diseño de las interacciones entre estos dos como predictores. ; por lo que identificamos las variables predictoras y de respuesta como X y y respectivamente.

```
# Load the data
load("GroundnutToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env, G & GE
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
LineG <- Line %**% Geno
LineXGenoXEnv <- model.matrix(~ 0 + LineG:Env)

# Predictor and Response Variables
X <- cbind(Env, LineG, LineXGenoXEnv)
y <- cbind(PhenoToy$PYPP, PhenoToy$YPH)
```

Posteriormente, realizamos cinco particiones aleatorias de conjunto de líneas, con 80% este conjunto para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función *cv_random* (con los parámetros por defecto). Además, creamos los data frames vacíos *PredictionsPPP*, *PredictionsYPH* e *Hyperparams* que servirán para guardar los valores observados y predicho en cada ambiente y posteriormente evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
# Random Partition Line
set.seed(2022)
# Unique Lines
GIDs <- unique(PhenoToy$Line)
folds <- cv_random(length(GIDs))
```

```
# Data frames that will contain the variables:
```

```
PredictionsPYPP <- data.frame()
```

```
PredictionsYPH <- data.frame()
```

```
Hyperparams <- data.frame()
```

Posteriormente, se seguirá el siguiente proceso **para cada partición y para cada variable de respuesta**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta, identificando primero las líneas correspondientes a este conjunto;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo valores entre 5 y 30 para el hiperparámetro *tree_number* y valores entre 5 y 15 para el hiperparámetro *node_size*, con "Optimización bayesiana" como tipo de tuneo. Cabe señalar que estos no son los únicos hiperparámetros ajustables en el modelo;
3. Con el modelo obtenido en (2), se predicen las variables de respuesta en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. Se crean los data frames *FoldPredictionsPYPP* y *FoldPredictionsYPH* que contienen las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba y para cada variable de respuesta respectiva.
 - b. Cada fila de *FoldPredictionPYPP* se agrega al data frame *PredictionsPYPP*; y cada fila de *FoldPredictionYPH* se agrega al data frame *PredictionsYPH*
5. Identificación de hiperparámetros:
 - a. *HyperparamsFold* que contiene las columnas *tree_number*, *node_size*, *multivariate_loss* y *Fold*, donde *multivariate_loss* es el costo del modelo para cada combinación de los hiperparámetros especificados y el número de pliegues.
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
6. Se muestran los hiperparámetros óptimos.

```
# Model training and predictions of the ith partition
```

```
for (i in seq_along(folds)) {  
  cat("*** Fold:", i, "***\n")
```

```
  # Identify the training and testing Line sets
```

```
  fold <- folds[[i]]
```

```
  Lines_sam_i <- GIDs[fold$training]
```

```
  fold_i <- which(PhenoToy$Line %in% Lines_sam_i)
```

```

# Identify the training and testing sets
X_training <- X[fold_i, ]
X_testing <- X[-fold_i, ]
y_training <- y[fold_i, ]
y_testing <- y[-fold_i, ]

# Model training
model <- random_forest(
  x = X_training,
  y = y_training,

  # Specify the hyperparameter ranges
  trees_number = list(min = 5, max = 30),
  node_size = list(min = 5, max = 15),
  tune_type = "Bayesian_optimization",
  tune_bayes_samples_number = 5,
  tune_bayes_iterations_number = 5,

  # In this example the iterations wont be shown
  verbose = FALSE
)

# Testing Predictions
predictions <- predict(model, X_testing)

# Predictions of PYPP for the Fold
PredictionsPYPP <- data.frame(
  Fold = i,
  Line = PhenoToy$Line[-fold_i],
  Env = PhenoToy$Env[-fold_i],
  Observed = y_testing[, 1],
  Predicted = predictions$V1$predicted
)
PredictionsPYPP <- rbind(PredictionsPYPP, FoldPredictionsPYPP)

# Predictions of YPH for the Fold
FoldPredictionsYPH <- data.frame(
  Fold = i,
  Line = PhenoToy$Line[-fold_i],
  Env = PhenoToy$Env[-fold_i],
  Observed = y_testing[, 2],
  Predicted = predictions$V2$predicted
)
PredictionsYPH <- rbind(PredictionsYPH, FoldPredictionsYPH)

# Hyperparams
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)

```



```

Hyperparams <- rbind(Hyperparams, HyperparamsFold)

# Best hyperparams of the model
cat("*** Optimal hyperparameters: ***\n")
print(model$best_hyperparams)
}
#> *** Fold: 1 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 18
#>
#> $node_size
#> [1] 8
#>
#> $multivariate_loss
#> [1] 0.3597012
#>
#> *** Fold: 2 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 24
#>
#> $node_size
#> [1] 6
#>
#> $multivariate_loss
#> [1] 0.3398092
#>
#> *** Fold: 3 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 17
#>
#> $node_size
#> [1] 7
#>
#> $multivariate_loss
#> [1] 0.3422837
#>
#> *** Fold: 4 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 30
#>
#> $node_size
#> [1] 8
#>
#> $multivariate_loss
#> [1] 0.3508043
#>

```

```

#> *** Fold: 5 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 15
#>
#> $node_size
#> [1] 9
#>
#> $multivariate_loss
#> [1] 0.3243171

```

Repitiendo este proceso para cada partición, los *data frames* *PredictionsPYPP* y *PredictionsYPH* contienen las columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición en su respectiva variable de respuesta, donde las predicciones se realizan eligiendo el óptimo. hiperparámetros (entre los posibles valores de estos) que minimizan la función de coste con el tipo de tuneo "Bayesian Optimization" correspondiente al formato necesario para utilizar la función *gs_summaries* sobre estas predicciones en el caso de variables continuas.

La función *gs_summaries* devuelve una lista que contiene tres *data frames* correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```

head(PredictionsPYPP)
#>   Fold   Line      Env Observed Predicted
#> 1     5 Gangapuri ALIYARNAGAR_R15     6.75  9.073130
#> 2     5 Gangapuri ICRISAT_PR15-16     7.20  6.432896
#> 3     5 Gangapuri ICRISAT_R15      8.26  8.297326
#> 4     5 Gangapuri JALGOAN_R15     6.10 10.175690
#> 5     5 ICG10036 ALIYARNAGAR_R15    13.20  9.258973
#> 6     5 ICG10036 ICRISAT_PR15-16     3.16  5.734787
unique(PredictionsPYPP$Fold)
#> [1] 5 7
head(PredictionsYPH)
#>   Fold   Line      Env Observed Predicted
#> 1     1 DTG15 ALIYARNAGAR_R15  1081.23 1098.2221
#> 2     1 DTG15 ICRISAT_PR15-16  2980.46 1416.4958
#> 3     1 DTG15 ICRISAT_R15    1282.71 1147.1809
#> 4     1 DTG15 JALGOAN_R15    1262.27 1305.8512
#> 5     1 ICG3746 ALIYARNAGAR_R15   559.92  971.6415
#> 6     1 ICG3746 ICRISAT_PR15-16 1672.00 1132.3552
unique(PredictionsYPH$Fold)
#> [1] 1 2 3 4 5

# Summaries
summariesPYPP <- gs_summaries(PredictionsPYPP)
summariesYPH <- gs_summaries(PredictionsYPH)

# Elements of summaries
names(summariesPYPP)
#> [1] "line" "env" "fold"

```

```

# Summaries by Line
head(summariesPYPP$line)
#>      Line Observed Predicted Difference
#> 1  CSMG84-1   9.9500    9.8150    0.1350
#> 2  ICG10036   8.6850    8.1453    0.5397
#> 3    DTG15  10.4400   11.0549    0.6149
#> 4    TG19    8.0317    8.9928    0.9612
#> 5 ICGV99085   8.9900   10.0123    1.0223
#> 6 ICGV00248  12.3800   11.1787    1.2013
head(summariesYPH$line)
#>      Line Observed Predicted Difference
#> 1 ICGV00248 1822.082  1784.794   37.2886
#> 2 ICGV07217 1446.125  1401.187   44.9377
#> 3    TG19 1217.993  1283.159   65.1661
#> 4  ICG10036 1050.182  1129.657   79.4740
#> 5 ICGV91114 1535.213  1421.521  113.6914
#> 6 Gangapuri 1140.013  1263.926  123.9135

# Summaries by Environment
summariesPYPP$env[, 1:8]
#>      Env      MSE MSE_SE  RMSE RMSE_SE  NRMSE
#> 1 ALIYARNAGAR_R15 10.1600 1.7707 3.1753  0.2788 1.1140
#> 2 ICRISAT_PR15-16 12.2790 1.7972 3.4947  0.2571 2.5687
#> 3   ICRISAT_R15   8.9848 0.4396 2.9966  0.0733 0.9544
#> 4   JALGOAN_R15  28.8130 5.0512 5.3469  0.4723 0.9941
#> 5      Global   9.2080 3.5897 2.9738  0.6036 1.0187
#>      NRMSE_SE      MAE
#> 1   0.1568 2.8602
#> 2   0.2752 3.2141
#> 3   0.0345 2.1569
#> 4   0.3195 4.4795
#> 5   0.1517 2.3379
summariesYPH$env[, 1:8]
#>      Env      MSE      MSE_SE      RMSE  RMSE_SE  NRMSE
#> 1 ALIYARNAGAR_R15 347220.3  71393.06 577.7885  57.8377 0.8146
#> 2 ICRISAT_PR15-16 471056.2 120472.32 661.5526  91.3844 1.1639
#> 3   ICRISAT_R15 295625.9 117648.96 498.6586 108.3576 0.7345
#> 4   JALGOAN_R15 577924.9 301603.56 682.3551 167.5683 1.0197
#> 5      Global 180671.5  69393.14 398.4810  73.9670 0.8885
#>      NRMSE_SE      MAE
#> 1   0.0710 489.7737
#> 2   0.1791 568.3065
#> 3   0.0672 388.0073
#> 4   0.0785 548.1157
#> 5   0.1006 297.5453

# Summaries by Fold
summariesPYPP$fold[, 1:8]
#>      Fold      MSE MSE_SE  RMSE RMSE_SE  NRMSE NRMSE_SE      MAE
#> 1      5 14.5785 3.2659 3.7509  0.4119 1.3711  0.3200 3.2026

```

```
#> 2      7 15.5399 6.1230 3.7558 0.6913 1.4445 0.4821 3.1528
#> 3 Global 9.2080 3.5897 2.9738 0.6036 1.0187 0.1517 2.3379
summariesYPH$fold[, 1:8]
#>      Fold      MSE  MSE_SE      RMSE RMSE_SE  NRMSE NRMSE_SE
#> 1      1 342153.9 135612.27 538.6147 131.7170 0.8589 0.1230
#> 2      2 264723.1 43956.03 509.4492 41.5717 0.9090 0.0495
#> 3      3 892208.5 317800.28 897.7897 169.4916 0.9023 0.1246
#> 4      4 252965.1 65033.21 489.9950 65.4980 0.7894 0.0613
#> 5      5 362733.6 84644.91 589.5948 70.9732 1.2062 0.2372
#> 6 Global 180671.5 69393.14 398.4810 73.9670 0.8885 0.1006
#>      MAE
#> 1 423.5188
#> 2 452.2880
#> 3 708.3477
#> 4 410.1784
#> 5 498.4210
#> 6 297.5453
```

Además, Hyperparams contiene las columnas *tree_number*, *node_size*, *multivariate_loss* y *Fold*, donde el valor de *multivariate_loss* corresponde al costo del modelo para cada combinación de partición y valores de hiperparámetro especificados, ordenados de menor a mayor dentro de cada partición.

```
# First rows of Hyperparams
head(Hyperparams)
#>      trees_number node_size multivariate_loss Fold
#> 4              18         8         0.3597012    1
#> 7              19         7         0.3613473    1
#> 8              16         7         0.3614817    1
#> 3              22        11         0.3632946    1
#> 10             23        14         0.3678480    1
#> 1              20        11         0.3682799    1
# Last rows of Hyperparams
tail(Hyperparams)
#>      trees_number node_size multivariate_loss Fold
#> 24              26        12         0.3309550    5
#> 44              15        10         0.3317055    5
#> 64              28         6         0.3338139    5
#> 14              16        12         0.3402441    5
#> 94              13         9         0.3423154    5
#> 34              22        12         0.3432523    5
```

5.5 Ejemplo multivariado mixto (con respuestas binaria, categórica y continua) Con optimización bayesiana con validación cruzada 7-fold con *Env+G\$ en el predictor

Este ejemplo evalúa un modelo Random Forest con validación cruzada de 7 veces, para una respuesta continua, una respuesta binaria y una respuesta categórica, usando el efecto

Environment y la matriz G como predictores, además de usa la "Optimización Bayesiana" como tipo de ajuste para hiperparámetros.

En este ejemplo, el conjunto de datos utilizado es *MaizeToy* y buscamos predecir las variables continuas *Yield* del data frame y_{bin} *PhenoToy*, y y_{cat} (que son transformaciones de las variables *ASI* y *PH* de *PhenoToy* respectivamente, con la ayuda de la función *ntile*) usando la matriz diseño de la variable *PhenoToy* Env y la matriz como G predictores; por lo que identificamos las variables predictoras y de respuesta como X y y respectivamente.

```
# Load the dataset
load("MaizeToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env & G
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %%% Geno

# Predictor and Response Variables
X <- cbind(Env, LineG)
y_bin <- BurStMisc::ntile(PhenoToy$PH, 2, result = "factor")
#> Warning in BurStMisc::ntile(PhenoToy$PH, 2, result = "factor"):
#> common values across groups: 1, 2
y_cat <- BurStMisc::ntile(PhenoToy$ASI, 4, result = "factor")
#> Warning in BurStMisc::ntile(PhenoToy$ASI, 4, result =
#> "factor"): common values across groups: 1, 2, 3, 4
y <- data.frame(PhenoToy$Yield, y_bin, y_cat)
```

Posteriormente realizamos las particiones correspondientes a CV de 7 veces, con la ayuda de la función *cv_kfold*. Además, creamos los data frames vacíos *PredictionsYield*, *PredictionsY_bin*, *PredictionsY_cat** e *Hypeparams* que servirán para guardar los valores observados y predichos en cada ambiente y posteriormente evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
# Set seed for reproducible results
set.seed(2022)
folds <- cv_kfold(records_number = nrow(X), k = 7)

# Data frames that will contain the variables:
PredictionsYield <- data.frame()
PredictionsY_bin <- data.frame()
PredictionsY_cat <- data.frame()
Hyperparams <- data.frame()
```

Posteriormente, se seguirá el siguiente proceso **para cada partición y para cada variable de respuesta**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo valores entre 5 y 30 para el hiperparámetro *tree_number* y valores entre 5 y 25 para el hiperparámetro *node_size*, con "Optimización bayesiana" como tipo de tuneo. Cabe señalar que estos no son los únicos hiperparámetros ajustables en el modelo;
3. Con el modelo obtenido en (2), se predicen las variables de respuesta en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. Se crean los data frames *FoldPredictionsYield*, *FoldPredictionsY_bin* y *FoldPredictionsY_cat* que contienen las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba y para cada variable de respuesta respectiva. Además, para el caso de *FoldPredictionsY_bin* y *FoldPredictionsY_cat*, también tienen columnas extra correspondientes a las probabilidades de que cada elemento pertenezca a cada categoría.
 - b. Cada fila de *FoldPredictionYield* se agrega al data frame de *PredictionsYield*; cada fila de *FoldPredictionY_bin* se agrega al data frame *PredictionsY_bin*; y cada fila de *FoldPredictionY_cat* se agrega al data frame *PredictionsY_cat*.
5. Identificación de hiperparámetros:
 - a. *HyperparamsFold* que contiene las columnas *tree_number*, *node_size*, *multivariate_loss* y *Fold*, donde *multivariate_loss* es el costo del modelo para cada combinación de los hiperparámetros especificados y el número de pliegues.
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
6. Se muestran los hiperparámetros óptimos.

```
# Model training and predictions of the ith partition
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")
  fold <- folds[[i]]

  # Identify the training and testing sets
  X_training <- X[fold$training, ]
  X_testing <- X[fold$testing, ]
  y_training <- y[fold$training, ]
  y_testing <- y[fold$testing, ]
}
```

```

# Model training
model <- random_forest(
  x = X_training,
  y = y_training,

  # Specify the hyperparameter ranges
  trees_number = list(min = 5, max = 30),
  node_size = list(min = 5, max = 25),
  tune_type = "Bayesian_optimization",
  tune_bayes_samples_number = 5,
  tune_bayes_iterations_number = 5
)

# Testing Predictions
predictions <- predict(model, X_testing)

# Predictions of Yield for the Fold
FoldPredictionsYield <- data.frame(
  Fold = i,
  Line = PhenoToy$Line[fold$testing],
  Env = PhenoToy$Env[fold$testing],
  Observed = y_testing[, 1],
  Predicted = predictions$PhenoToy.Yield$predicted
)
PredictionsYield <- rbind(PredictionsYield, FoldPredictionsYield)

# Predictions of Y_bin for the Fold
FoldPredictionsY_bin <- cbind(
  data.frame(
    Fold = i,
    Line = PhenoToy$Line[fold$testing],
    Env = PhenoToy$Env[fold$testing],
    Observed = y_testing[, 2],
    Predicted = predictions$y_bin$predicted
  ),
  predictions$y_bin$probabilities
)
PredictionsY_bin <- rbind(PredictionsY_bin, FoldPredictionsY_bin)

# Predictions of Y_cat for the Fold
FoldPredictionsY_cat <- cbind(
  data.frame(
    Fold = i,
    Line = PhenoToy$Line[fold$testing],
    Env = PhenoToy$Env[fold$testing],
    Observed = y_testing[, 3],
    Predicted = predictions$y_cat$predicted
  ),
  predictions$y_cat$probabilities
)

```

```

)
PredictionsY_cat <- rbind(PredictionsY_cat, FoldPredictionsY_cat)

# Hyperparams
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)

# Best hyperparams of the model
cat("*** Optimal hyperparameters: ***\n")
print(model$best_hyperparams)
}
#> *** Fold: 1 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1 / 10
#> KFoldCV: 1 / 5
#> KFoldCV: 2 / 5
#> KFoldCV: 3 / 5
#> KFoldCV: 4 / 5
#> KFoldCV: 5 / 5
#> Combination: 2 / 10
#> KFoldCV: 1 / 5
#> KFoldCV: 2 / 5
#> KFoldCV: 3 / 5
#> KFoldCV: 4 / 5
#> KFoldCV: 5 / 5
#> Combination: 3 / 10
#> KFoldCV: 1 / 5
#> KFoldCV: 2 / 5
#> KFoldCV: 3 / 5
#> KFoldCV: 4 / 5
#> KFoldCV: 5 / 5
#> Combination: 4 / 10
#> KFoldCV: 1 / 5
#> KFoldCV: 2 / 5
#> KFoldCV: 3 / 5
#> KFoldCV: 4 / 5
#> KFoldCV: 5 / 5
#> Combination: 5 / 10
#> KFoldCV: 1 / 5
#> KFoldCV: 2 / 5
#> KFoldCV: 3 / 5
#> KFoldCV: 4 / 5
#> KFoldCV: 5 / 5
#> Combination: 6 / 10
#> KFoldCV: 1 / 5
#> KFoldCV: 2 / 5
#> KFoldCV: 3 / 5
#> KFoldCV: 4 / 5
#> KFoldCV: 5 / 5

```



```

#> Combination: 7 / 10
#> KFoldCV: 1 / 5
#> KFoldCV: 2 / 5
#> KFoldCV: 3 / 5
#> KFoldCV: 4 / 5
#> KFoldCV: 5 / 5
#> Combination: 8 / 10
#> KFoldCV: 1 / 5
#> KFoldCV: 2 / 5
#> KFoldCV: 3 / 5
#> KFoldCV: 4 / 5
#> KFoldCV: 5 / 5
#> Combination: 9 / 10
#> KFoldCV: 1 / 5
#> KFoldCV: 2 / 5
#> KFoldCV: 3 / 5
#> KFoldCV: 4 / 5
#> KFoldCV: 5 / 5
#> Combination: 10 / 10
#> KFoldCV: 1 / 5
#> KFoldCV: 2 / 5
#> KFoldCV: 3 / 5
#> KFoldCV: 4 / 5
#> KFoldCV: 5 / 5
#> *** Fitting Multivariate Random Forest model ***
#> *** Model evaluation completed in 14.4296 secs ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 21
#>
#> $node_size
#> [1] 11
#>
#> $multivariate_loss
#> [1] 0.3086651
#>
#> *** Fold: 2 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1 / 10
#> KFoldCV: 1 / 5
#> KFoldCV: 2 / 5
#> KFoldCV: 3 / 5
#> KFoldCV: 4 / 5
#> KFoldCV: 5 / 5
#> Combination: 2 / 10
#> KFoldCV: 1 / 5
#> KFoldCV: 2 / 5
#> KFoldCV: 3 / 5
#> KFoldCV: 4 / 5
#> KFoldCV: 5 / 5

```

```

#> Combination: 3 / 10
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 4 / 10
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 5 / 10
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 6 / 10
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 7 / 10
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 8 / 10
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 9 / 10
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 10 / 10
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> *** Fitting Multivariate Random Forest model ***
#> *** Model evaluation completed in 13.8351 secs ***

```

```

#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 30
#>
#> $node_size
#> [1] 5
#>
#> $multivariate_loss
#> [1] 0.3254175
#>
#> *** Fold: 3 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1 / 10
#>   KFoldCV: 1 / 5
#>   KFoldCV: 2 / 5
#>   KFoldCV: 3 / 5
#>   KFoldCV: 4 / 5
#>   KFoldCV: 5 / 5
#> Combination: 2 / 10
#>   KFoldCV: 1 / 5
#>   KFoldCV: 2 / 5
#>   KFoldCV: 3 / 5
#>   KFoldCV: 4 / 5
#>   KFoldCV: 5 / 5
#> Combination: 3 / 10
#>   KFoldCV: 1 / 5
#>   KFoldCV: 2 / 5
#>   KFoldCV: 3 / 5
#>   KFoldCV: 4 / 5
#>   KFoldCV: 5 / 5
#> Combination: 4 / 10
#>   KFoldCV: 1 / 5
#>   KFoldCV: 2 / 5
#>   KFoldCV: 3 / 5
#>   KFoldCV: 4 / 5
#>   KFoldCV: 5 / 5
#> Combination: 5 / 10
#>   KFoldCV: 1 / 5
#>   KFoldCV: 2 / 5
#>   KFoldCV: 3 / 5
#>   KFoldCV: 4 / 5
#>   KFoldCV: 5 / 5
#> Combination: 6 / 10
#>   KFoldCV: 1 / 5
#>   KFoldCV: 2 / 5
#>   KFoldCV: 3 / 5
#>   KFoldCV: 4 / 5
#>   KFoldCV: 5 / 5
#> Combination: 7 / 10
#>   KFoldCV: 1 / 5

```

```

#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 8 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 9 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 10 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> *** Fitting Multivariate Random Forest model ***
#> *** Model evaluation completed in 13.1847 secs ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 30
#>
#> $node_size
#> [1] 5
#>
#> $multivariate_loss
#> [1] 0.3283907
#>
#> *** Fold: 4 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 2 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 3 / 10
#>      KFoldCV: 1 / 5

```

```

#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 4 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 5 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 6 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 7 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 8 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 9 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 10 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> *** Fitting Multivariate Random Forest model ***
#> *** Model evaluation completed in 13.6239 secs ***
#> *** Optimal hyperparameters: ***
#> $trees_number

```

```

#> [1] 22
#>
#> $node_size
#> [1] 5
#>
#> $multivariate_loss
#> [1] 0.360276
#>
#> *** Fold: 5 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1 / 10
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 2 / 10
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 3 / 10
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 4 / 10
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 5 / 10
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 6 / 10
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5
#>     KFoldCV: 4 / 5
#>     KFoldCV: 5 / 5
#> Combination: 7 / 10
#>     KFoldCV: 1 / 5
#>     KFoldCV: 2 / 5
#>     KFoldCV: 3 / 5

```

```

#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 8 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 9 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 10 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> *** Fitting Multivariate Random Forest model ***
#> *** Model evaluation completed in 12.6999 secs ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 16
#>
#> $node_size
#> [1] 16
#>
#> $multivariate_loss
#> [1] 0.3113797
#>
#> *** Fold: 6 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 2 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 3 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5

```

```

#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 4 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 5 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 6 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 7 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 8 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 9 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 10 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> *** Fitting Multivariate Random Forest model ***
#> *** Model evaluation completed in 14.9861 secs ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 30
#>

```



```

#> $node_size
#> [1] 8
#>
#> $multivariate_loss
#> [1] 0.303435
#>
#> *** Fold: 7 ***
#> *** Bayesian Optimization Tuning ***
#> Combination: 1 / 10
#>   KFoldCV: 1 / 5
#>   KFoldCV: 2 / 5
#>   KFoldCV: 3 / 5
#>   KFoldCV: 4 / 5
#>   KFoldCV: 5 / 5
#> Combination: 2 / 10
#>   KFoldCV: 1 / 5
#>   KFoldCV: 2 / 5
#>   KFoldCV: 3 / 5
#>   KFoldCV: 4 / 5
#>   KFoldCV: 5 / 5
#> Combination: 3 / 10
#>   KFoldCV: 1 / 5
#>   KFoldCV: 2 / 5
#>   KFoldCV: 3 / 5
#>   KFoldCV: 4 / 5
#>   KFoldCV: 5 / 5
#> Combination: 4 / 10
#>   KFoldCV: 1 / 5
#>   KFoldCV: 2 / 5
#>   KFoldCV: 3 / 5
#>   KFoldCV: 4 / 5
#>   KFoldCV: 5 / 5
#> Combination: 5 / 10
#>   KFoldCV: 1 / 5
#>   KFoldCV: 2 / 5
#>   KFoldCV: 3 / 5
#>   KFoldCV: 4 / 5
#>   KFoldCV: 5 / 5
#> Combination: 6 / 10
#>   KFoldCV: 1 / 5
#>   KFoldCV: 2 / 5
#>   KFoldCV: 3 / 5
#>   KFoldCV: 4 / 5
#>   KFoldCV: 5 / 5
#> Combination: 7 / 10
#>   KFoldCV: 1 / 5
#>   KFoldCV: 2 / 5
#>   KFoldCV: 3 / 5
#>   KFoldCV: 4 / 5
#>   KFoldCV: 5 / 5

```

```

#> Combination: 8 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 9 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> Combination: 10 / 10
#>      KFoldCV: 1 / 5
#>      KFoldCV: 2 / 5
#>      KFoldCV: 3 / 5
#>      KFoldCV: 4 / 5
#>      KFoldCV: 5 / 5
#> *** Fitting Multivariate Random Forest model ***
#> *** Model evaluation completed in 14.1194 secs ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 20
#>
#> $node_size
#> [1] 9
#>
#> $multivariate_loss
#> [1] 0.3638747

```

PredictionsYield contiene las columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* ; el data frame *PredictionsY_bin* contiene las columnas *Fold*, *Line*, *Env*, *Observed*, *Predicted* , 1 y 2 ; y el data frame *PredictionsY_cat* contiene las columnas *Fold*, *Line*, *Env*, *Observed*, *Predicted*, 1, 2 , 3 y 4, todo eso para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo los hiperparámetros óptimos (entre sus posibles valores) que minimizan la función de coste con el tipo de tuneo "Bayesian Optimization", correspondiente al formato necesario para utilizar la función *gs_summaries* sobre estas predicciones en el caso de variables continuas, binarias y categóricas, respectivamente.

La función *gs_summaries* devuelve una lista que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```

head(PredictionsYield)
#>      Fold      Line Env Observed Predicted
#> 1      1 CKDHL0032 KTI      6.24  5.922975
#> 2      1 CKDHL0049 EBU      4.72  6.252130
#> 3      1 CKDHL0050 EBU      4.98  6.288283
#> 4      1 CKDHL0052 KTI      7.20  6.002546
#> 5      1 CKDHL0085 KTI      7.41  6.143315
#> 6      1 CKDHL0097 KTI      4.45  5.893860

```

```

unique(PredictionsYield$Fold)
#> [1] 1 2 3 4 5 6 7
head(PredictionsY_bin)
#>   Fold      Line Env Observed Predicted      1      2
#> 1    1 CKDHL0032 KTI      1      2 0.3818485 0.6181515
#> 2    1 CKDHL0049 EBU      2      2 0.3401469 0.6598531
#> 3    1 CKDHL0050 EBU      2      2 0.3615337 0.6384663
#> 4    1 CKDHL0052 KTI      2      2 0.3557111 0.6442889
#> 5    1 CKDHL0085 KTI      2      2 0.3096140 0.6903860
#> 6    1 CKDHL0097 KTI      1      2 0.3860942 0.6139058
unique(PredictionsY_bin$Fold)
#> [1] 1 2 3 4 5 6 7

# Summaries
summariesYield <- gs_summaries(PredictionsYield)
summariesY_bin <- gs_summaries(PredictionsY_bin)
summariesY_cat <- gs_summaries(PredictionsY_cat)

# Elements of summaries
names(summariesYield)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summariesYield$line)
#>      Line Observed Predicted Difference
#> 1 CKDHL0054   5.9233    5.9612    0.0378
#> 2 CKDHL0136   5.8800    5.9227    0.0427
#> 3 CKDHL0027   5.8800    5.8289    0.0511
#> 4 CKDHL0515   6.0100    5.9523    0.0577
#> 5 CKDHL0203   5.8267    5.7633    0.0633
#> 6 CKDHL0474   6.1933    6.2762    0.0828
head(summariesY_bin$line)
#>      Line Observed Predicted      X1      X2
#> 1 CKDHL0027      1      2 0.5520 0.4480
#> 2 CKDHL0032      1      2 0.5450 0.4550
#> 3 CKDHL0046      1      2 0.5442 0.4558
#> 4 CKDHL0049      2      2 0.5206 0.4794
#> 5 CKDHL0050      2      2 0.4977 0.5023
#> 6 CKDHL0052      1      2 0.4540 0.5460
head(summariesY_cat$line)
#>      Line Observed Predicted      X1      X2      X3      X4
#> 1 CKDHL0027      1      2 0.2173 0.2275 0.3235 0.2316
#> 2 CKDHL0032      1      4 0.2328 0.2513 0.2697 0.2462
#> 3 CKDHL0046      4      2 0.2195 0.2680 0.2811 0.2314
#> 4 CKDHL0049      3      2 0.2295 0.2454 0.2921 0.2330
#> 5 CKDHL0050      1      1 0.2358 0.2241 0.2587 0.2814
#> 6 CKDHL0052      2      4 0.2372 0.2396 0.2597 0.2635

# Summaries by Environment
summariesYield$env[, 1:9]
#>      Env      MSE MSE_SE      RMSE RMSE_SE      NRMSE NRMSE_SE      MAE

```

```

#> 1    EBU 0.6075 0.1535 0.7417 0.0978 1.8133 0.4550 0.6453
#> 2    KAK 0.5715 0.1960 0.6405 0.1639 1.1792 0.1526 0.5187
#> 3    KTI 1.2564 0.2967 1.0417 0.1689 1.3283 0.4404 0.9225
#> 4 Global 0.8215 0.1553 0.8767 0.0938 0.9368 0.0469 0.7138
#>    MAE_SE
#> 1 0.0944
#> 2 0.1318
#> 3 0.1475
#> 4 0.0891

# Summaries by Fold
summariesYield$fold[, 1:8]
#>      Fold    MSE MSE_SE    RMSE RMSE_SE    NRMSE NRMSE_SE    MAE
#> 1      1 0.8559 0.4181 0.8007 0.3277 0.9758 0.0342 0.7330
#> 2      2 0.7603 0.3396 0.8220 0.2057 1.8774 0.8356 0.6894
#> 3      3 1.3995 0.4063 1.1579 0.1714 1.8279 0.8626 0.9913
#> 4      4 0.1647 0.1331 0.3276 0.1694 0.8518 0.2732 0.2640
#> 5      5 1.1402 0.5154 1.0052 0.2548 1.1110 0.3129 0.8219
#> 6      6 0.5848 0.3978 0.6760 0.2528 1.1295 0.0331 0.5763
#> 7      7 0.7771 0.1874 0.8665 0.1144 2.2407 0.8563 0.7927
#> 8 Global 0.8215 0.1553 0.8767 0.0938 0.9368 0.0469 0.7138

```

Además, Hyperparams contiene las columnas *tree_number*, *node_size*, *multivariate_loss* y *Fold*, donde el valor de *multivariate_loss* corresponde al costo del modelo para cada combinación de partición y valores de hiperparámetro especificados, ordenados de menor a mayor dentro de cada partición.

```

# First rows of Hyperparams
head(Hyperparams)
#>   trees_number node_size multivariate_loss Fold
#> 2             21        11         0.3086651    1
#> 6             28         9         0.3158732    1
#> 8             30         5         0.3230845    1
#> 3             24        12         0.3298924    1
#> 10            22         9         0.3349999    1
#> 5              8        13         0.3515807    1

# Last rows of Hyperparams
tail(Hyperparams)
#>   trees_number node_size multivariate_loss Fold
#> 86            30         5         0.3991094    7
#> 96            30        12         0.4025800    7
#> 26            13        16         0.4067875    7
#> 16            17        25         0.5216477    7
#> 56            24        20         0.5217156    7
#> 46            12        23         0.5218220    7

```

5.6 Ejemplo con kernels con Grid search y particiones aleatorias

Este ejemplo evalúa un modelo Random Forest con cinco particiones aleatorias, con 20% los datos par el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de cada partición (los parámetros predeterminados de la unción `cv_random`), para una respuesta continua, utilizando la matriz de diseño de PhenoToy Env variable, la matriz descrita *G* anteriormente y la matriz de diseño de la interacción entre estas dos, como predictores; así como el uso de "Grid Search" como un tipo de tuneo para los hiperparámetros *tree_number*, *sampled_x_vars_number* y *node_size*. Todo esto para los tipos de Kernel: "Linear", "Polinomial", "Sigmoid", "Gaussian", "Exponential", "Arc_cosine" y "Arc_cosine_L".

En este ejemplo, el conjunto de datos utilizado es *EYTTToy* y el objetivo es predecir la variable continua *GY* del data frame *PhenoToy* utilizando la matriz de diseño de la variable PhenoToy Env, la matriz descrita *G* anteriormente y la matriz de diseño de la interacción entre estos dos, como predictores; por lo que identificamos las variables predictoras y de respuesta como *X* y *y* respectivamente.

```
# Load the data
load("EYTTToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env, G & GE
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LinexGeno <- Line %*% Geno
LinexGenoxEnv <- model.matrix(~ 0 + LinexGeno:Env)

# Predictor and Response Variables
X <- cbind(Env, LinexGeno, LinexGenoxEnv)
y <- PhenoToy$GY

dim(X)
#> [1] 120 154
print(y[1:7])
#> [1] 5.510785 6.087132 6.754944 2.752278 6.399115 5.951386
#> [7] 6.109080
typeof(y)
#> [1] "double"
```

Tenga en cuenta que la variable de respuesta *y* es una variable continua (un vector con elementos de tipo "doble"), lo cual es importante para que el modelo se entrene automáticamente para una variable continua.

A diferencia de los ejmplos anteriores, ahora buscamos evaluar el modelo para cada tipo de kernel mencionado anteriormente. Por ello, creamos un vector en el que indicamos los tipos de kernel que queremos aplicar a la matriz X Además, creamos las listas vacías *PredictionsAll*, *TimesAll*, *HyperparamsAll* y *SummariesAll* que se utilizarán para guardar las predicciones, los tiempos de ejecución, los hiperparámetros y los resúmenes de cada modelo entrenado, es decir, para cada tipo de kernel; lo que a su vez servirá para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
kernels <- c(
  "linear",
  "polynomial",
  "sigmoid",
  "Gaussian",
  "exponential",
  "arc_cosine",
  "Arc_cosine_L"
)

# Empty lists that will contain Predictions,
# Times of execution & Summaries for each type of kernel
PredictionsAll <- list()
TimesAll <- list()
HyperparamsAll <- list()
SummariesAll <- list()
```

Posteriormente, se seguirá el siguiente proceso **para cada tipo de Kernel**:

1. identifique la variable *arc_deep* con el valor 2. Si el tipo de Kernel es "Arc_cosine_L", el valor de la variable *arc_deep* se cambia a 3 y el *kernel_type* se identifica como "Arc_cosine"; de lo contrario, *kernel_type* se identifica como el kernel predeterminado.
2. El tipo de kernel establecido en (1) se aplica a la matriz de datos X , asignando al argumento *arc_cosine_deep* el valor establecido en la variable *arc_deep*. Tenga en cuenta que el argumento *arc_cosine_deep* se ignora si el tipo de kernel no es *Arc_cosine*.
3. Luego realizamos cinco particiones aleatorias, con 80% los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función *cv_random*.
4. Predictions, *Times* e *Hyperparams* que se utilizarán para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función *gs_summaries*, además de guardar los tiempos de ejecución de cada modelo entrenado para cada partición.
5. **Para cada partición:**
 1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;

2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo los valores 100, 300 y 500 para el hiperparámetro *tree_number*, los valores 0.3, 0.5 y 0.8 para el hiperparámetro *sampled_x_vars_number* y los valores 5 y 10 para el hiperparámetro *node_size*, con "Grid Search" como tipo de tuneo (parámetro por defecto de *tune_type*);
 3. Con el modelo obtenido en (2), se predice la variable respuesta y en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
 4. Identificación de las predicciones del conjunto de prueba: Se crea el data frame *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba. Además, cada fila de *FoldPrediction* se agrega al data frame *Predictions*.
 5. Identificación del tiempo de ejecución del entrenamiento del modelo obtenido en (2): Se crea el data frame *FoldTime* que contiene la columna que especifica el tipo de *Kernel* utilizado para entrenar el modelo, el número de *Fold* y la columna *Minutos* que indica el tiempo de ejecución, en minutos, del entrenamiento del modelo obtenido en (2). Además, cada fila de *FoldTime* se agrega al data frame *Times*.
 6. Identificación de hiperparámetros; Se crea el data frame *HyperparamsFold* que contiene las columnas *tree_number*, *node_size*, *sampled_x_vars_number*, *mse* y *Fold*, donde *m* es el costo del modelo para cada combinación de los hiperparámetros especificados y el número de *Fold*. Además, cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
- data frame de predicciones contiene columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba para cada partición, donde las predicciones se realizan eligiendo los hiperparámetros óptimos (entre los posibles valores especificados) que minimizan la función de costo con el "Grid Search", correspondiente al formato necesario para utilizar la función *gs_summaries* sobre estas predicciones en el caso de variables continuas.
6. *Predictions* y con la ayuda de la función *gs_summaries*, evaluamos la capacidad predictiva del modelo entrenado para el tipo de kernel especificado. La función *gs_summaries* devuelve una lista que identificamos como *summaries* y que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.
 7. Finalmente, se crea un elemento con el nombre de kernel especificado en cada una de las listas *PredictionsAll*, *TimesAll*, *HyperparamsAll* y *SummariesAll*, que corresponden a los marcos de datos de las listas *Predictions*, *Times*, *Hyperparams* y *summaries*, respectivamente.

```
for (kernel in kernels) {
  cat("*** Kernel:", kernel, "***\n")

  # Identify the arc_deep and the kernel
```

```

arc_deep <- 2
if (kernel == "Arc_cosine_L") {
  arc_deep <- 3
  kernel <- "arc_cosine"
} else {
  kernel <- kernel
}

# Compute the kernel
X <- kernelize(X, kernel = kernel, arc_cosine_deep = arc_deep)

# Random Partition
set.seed(2022)
folds <- cv_random(
  records_number = nrow(X),
  folds_number = 5,
  testing_proportion = 0.2
)

# Empty data frames that will contain Predictions, Times
# of execution & Summaries for each partition
Predictions <- data.frame()
Times <- data.frame()
Hyperparams <- data.frame()

for (i in seq_along(folds)) {
  cat("\t*** Fold:", i, "****\n")
  fold <- folds[[i]]

  # Identify the training and testing sets
  X_training <- X[fold$training, ]
  X_testing <- X[fold$testing, ]
  y_training <- y[fold$training]
  y_testing <- y[fold$testing]

  # Model training
  model <- random_forest(
    x = X_training,
    y = y_training,

    # Specify the hyperparameters values
    trees_number = c(100, 300, 500),
    sampled_x_vars_number = c(0.3, 0.5, 0.8),
    node_size = c(5, 10),
    tune_type = "grid_search",
    tune_grid_proportion = 0.8,
    # In this example the iterations wont be shown
    verbose = FALSE
  )
}

```



```

# Testing Predictions
predictions <- predict(model, X_testing)

# Predictions for the Fold Fold
FoldPredictions <- data.frame(
  Fold = i,
  Line = PhenoToy$Line[fold$testing],
  Env = PhenoToy$Env[fold$testing],
  Observed = y_testing,
  Predicted = predictions$predicted
)
Predictions <- rbind(Predictions, FoldPredictions)

# Execution times
FoldTime <- data.frame(
  kernel = kernel,
  Fold = i,
  Minutes = as.numeric(model$execution_time, units = "mins")
)
Times <- rbind(Times, FoldTime)

# Hyperparams for the Fold
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)
}
# Summaries of the Folds
summaries <- gs_summaries(Predictions)

# Predictions, Times of execution & Summaries for the
# specified Kernel
PredictionsAll[[kernel]] <- Predictions
TimesAll[[kernel]] <- Times
HyperparamsAll[[kernel]] <- Hyperparams
SummariesAll[[kernel]] <- summaries
}
#> *** Kernel: Linear ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: polynomial ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***

```

```

#> *** Kernel: sigmoid ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: Gaussian ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: exponential ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: arc_cosine ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: Arc_cosine_L ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***

```

Recordando que este proceso se realizó para cada tipo de kernel, cada una de las listas *PredictionsAll*, *TimesAll*, *HyperparamsAll* y *SummariesAll* contiene las predicciones, los tiempos de ejecución, las combinaciones e los hiperparámetros con su correspondiente costo y los resúmenes, respectivamente, para cada tipo de kernel apliado a la matriz de datos *X*. A modo de ejemplo, a continuación se muestran los resultados obtenidos para el tipo de kernel “Sigmoid”:

```

# Predictions for the Sigmoid Kernel
tail(PredictionsAll$sigmoid)
#>      Fold      Line      Env Observed Predicted
#> 115      5 GID7631195 FlatDrip 3.101994  2.912365
#> 116      5 GID7625985 Flat5IR  6.955977  6.255739
#> 117      5 GID7626584 Bed5IR   5.440434  5.983940
#> 118      5 GID7629552 Flat5IR  5.766224  5.374563
#> 119      5 GID7729805 Bed5IR   6.629760  6.304862
#> 120      5 GID7632527 Bed5IR   6.457017  6.019779
# Times of execution for the Sigmoid Kernel
TimesAll$sigmoid
#>      kernel Fold  Minutes

```

```

#> 1 sigmoid      1 0.1147623
#> 2 sigmoid      2 0.1138184
#> 3 sigmoid      3 0.1185036
#> 4 sigmoid      4 0.1232462
#> 5 sigmoid      5 0.1099385
# Elements of SummariesAll
names(SummariesAll)
#> [1] "linear"      "polynomial"  "sigmoid"     "Gaussian"
#> [5] "exponential" "arc_cosine"
# Elements of summaries for the Sigmoid Kernel
names(SummariesAll$sigmoid)
#> [1] "line" "env"  "fold"
# Summaries by Line
head(SummariesAll$sigmoid$line)
#>      Line Observed Predicted Difference
#> 1 GID7630553    5.9344    5.9566    0.0221
#> 2 GID7462121    6.1329    6.1581    0.0253
#> 3 GID7629600    5.1390    5.1132    0.0258
#> 4 GID7634730    5.5399    5.5017    0.0382
#> 5 GID7626446    5.0824    5.1397    0.0574
#> 6 GID7632666    5.6445    5.5464    0.0981

# Summaries by Environment
SummariesAll$sigmoid$env[, 1:8]
#>      Env      MSE MSE_SE  RMSE RMSE_SE  NRMSE NRMSE_SE  MAE
#> 1 Bed5IR 0.4417 0.0738 0.6536 0.0603 1.2337 0.0568 0.5775
#> 2 EHT 0.6872 0.1825 0.8027 0.1036 0.9434 0.2034 0.7166
#> 3 Flat5IR 0.6264 0.0807 0.7846 0.0520 1.3007 0.0841 0.6745
#> 4 FlatDrip 2.3350 0.2525 1.5197 0.0799 5.1196 0.6441 1.1923
#> 5 Global 0.9241 0.1544 0.9457 0.0862 0.5929 0.0497 0.7239
SummariesAll$sigmoid$env[, 9:15]
#>      MAE_SE      Cor Cor_SE Intercept Intercept_SE  Slope
#> 1 0.0503 0.3622 0.1704 2.6465 2.1586 0.5866
#> 2 0.0787 0.4215 0.2556 1.3709 2.3220 0.8174
#> 3 0.0596 0.6568 0.1150 2.7460 0.6378 0.6391
#> 4 0.0844 -0.2729 0.1876 3.0654 0.2373 -0.0975
#> 5 0.0495 0.8131 0.0325 -1.4640 0.6094 1.2578
#>      Slope_SE
#> 1 0.3587
#> 2 0.3925
#> 3 0.1178
#> 4 0.0631
#> 5 0.1029
SummariesAll$sigmoid$env[, 16:19]
#>      R2 R2_SE  MAAPE MAAPE_SE
#> 1 0.2473 0.1670 0.0913 0.0088
#> 2 0.4391 0.0631 0.1184 0.0128
#> 3 0.4842 0.1332 0.1033 0.0098
#> 4 0.2153 0.0694 0.3938 0.0305
#> 5 0.6653 0.0536 0.1620 0.0173

```

```

# Summaries by Fold
SummariesAll$sigmoid$fold[, 1:8]
#>      Fold      MSE MSE_SE  RMSE RMSE_SE  NRMSE NRMSE_SE  MAE
#> 1      1 0.9154 0.3039 0.9227 0.1461 1.9485 0.9545 0.8330
#> 2      2 1.1098 0.3972 0.9875 0.2119 2.2914 0.8663 0.8100
#> 3      3 0.9408 0.5290 0.8767 0.2396 2.2253 1.0164 0.6995
#> 4      4 1.1961 0.6802 0.9868 0.2722 1.6311 0.5886 0.8380
#> 5      5 0.9509 0.3732 0.9270 0.1747 2.6504 1.5725 0.7705
#> 6 Global 0.9241 0.1544 0.9457 0.0862 0.5929 0.0497 0.7239
SummariesAll$sigmoid$fold[, 9:15]
#>      MAE_SE      Cor Cor_SE Intercept Intercept_SE Slope Slope_SE
#> 1 0.1105 0.6577 0.1186 0.8515 1.1981 0.8045 0.2857
#> 2 0.1568 0.0927 0.3307 2.9301 3.1740 0.3909 0.5413
#> 3 0.1476 0.1535 0.2812 4.0266 0.6925 0.1935 0.1727
#> 4 0.2285 0.3281 0.2463 2.5914 1.1250 0.4461 0.2858
#> 5 0.1058 0.2275 0.3260 1.8864 1.7593 0.5969 0.3827
#> 6 0.0495 0.8131 0.0325 -1.4640 0.6094 1.2578 0.1029
SummariesAll$sigmoid$fold[, 16:19]
#>      R2 R2_SE MAAPE MAAPE_SE
#> 1 0.4747 0.1592 0.1808 0.0662
#> 2 0.3366 0.1960 0.1785 0.0724
#> 3 0.2607 0.0891 0.1595 0.0702
#> 4 0.2897 0.1328 0.1992 0.1029
#> 5 0.3706 0.1220 0.1655 0.0546
#> 6 0.6653 0.0536 0.1620 0.0173

```

Además, los elementos d la lista *HyperparamsAll* contienen las columnas *tree_number*, *node_size*, *sampled_x_number*, *mse* y *Fold*, dnde el valor de la columna *mse* corresponde al costo del modelo para cada combinación de los valores de partición e hiperparámetro, ordenados de menor a ayor . mayor dentro de cada partición.

```

# First rows of Hyperparams
head(HyperparamsAll$sigmoid)
#>      trees_number node_size sampled_x_vars_number      mse Fold
#> 8              300         5                  60 1.486428    1
#> 9              500         5                  60 1.488343    1
#> 1             100         5                  36 1.510207    1
#> 7             100         5                  60 1.514091    1
#> 3             500         5                  36 1.526728    1
#> 15            500         5                  96 1.545332    1

# Last rows of Hyperparams
tail(HyperparamsAll$sigmoid)
#>      trees_number node_size sampled_x_vars_number      mse Fold
#> 174            300         10                  96 1.631972    5
#> 44             100         10                  36 1.634050    5
#> 123            500         10                  60 1.665363    5
#> 183            500         10                  96 1.673345    5

```

#> 163	100	10	96 1.693188	5
#> 54	300	10	36 1.701055	5

5.7 Ejemplo con sparse Kernels con Grid search y particiones aleatorias

Este ejemplo evalúa un modelo Random Forest con cinco particiones aleatorias, con 20% los datos para el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de cada partición (los parámetros predeterminados de la función `cv_random`), para una respuesta continua, utilizando la matriz de diseño de PhenoToy Env variable, la matriz descrita *G* anteriormente y a matriz de diseño de la interacción entre estas dos, como predictores; así como el uso de "Grid Search" como un tipo de tuneo para los hiperparámetros *tree_number*, *sampled_x_vars_number* y *node_size*. Todo esto para los llamados "Sparse Kernel Methods", con las posibles combinaciones entre los tipos de Kernel "Sparse_Gaussian" y "Sparse_Arc_cosine" con las proporciones 0.5, 0.6, 0.7, 0.8, 0.9 y 1.

En este ejemplo, el conjunto de datos utilizado es *MaizeToy* y el objetivo es predecir la variable continua *Yield* del data frame *PhenoToy* utilizando la matriz de diseño de la variable PhenoToy Env, la matriz descrita *G* anteriormente y la matriz de diseño de la interacción entre estos dos, como predictores; por lo que identificamos las variables predictoras y de respuesta como *X* y *y* respectivamente.

```
# Load the dataset
load("MaizeToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env, G & GE
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LinexGeno <- Line %*% Geno
LinexGenoEnv <- model.matrix(~ 0 + LinexGeno:Env)

# Predictor and Response Variables
X <- cbind(Env, LinexGeno, LinexGenoEnv)
y <- PhenoToy$Yield

dim(X)
#> [1] 90 123
print(y[1:7])
#> [1] 6.11 6.21 5.32 6.62 5.60 6.24 5.24
typeof(y)
#> [1] "double"
```

Tenga en cuenta que la variable de respuesta *y* es una variable continua (un vector con elementos de tipo "doble"), lo cual es importante para que el modelo se entrene automáticamente para una variable continua.

A diferencia de los ejemplos anteriores, ahora buscamos evaluar el modelo para cada una de las posibles combinaciones entre los tipos de Kernel "Sparse_Gaussian" y "Sparse_Arc_cosine" con las proporciones 0.5, 0.6, 0.7, 0.8, 0.9, 1. Por eso creamos un vector llamado *kernels* en el que indicamos los tipos de kernels que queremos aplicar a los de la matriz *X* y otro vector llamado *lines_proportions*. Además, creamos las listas vacías *PredictionsAll*, *TimesAll*, *HyperparamsAll* y *SummariesAll* que se utilizarán para guardar las predicciones, los tiempos de ejecución, los hiperparámetros y los resúmenes de cada modelo entrenado, es decir, para cada combinación entre tipo de kernel y proporción de líneas utilizadas, que a su vez servirán para guardar los valores observados y predichos en cada ambiente y para posteriormente evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
kernels <- c("Sparse_Gaussian", "Sparse_Arc_cosine")
lines_proportions <- c(0.5, 0.6, 0.7, 0.8, 0.9, 1)

# Empty lists that will contain Predictions,
# Times of execution & Summaries for each type of kernel
PredictionsAll <- list()
TimesAll <- list()
HyperparamsAll <- list()
SummariesAll <- list()
```

Posteriormente, se seguirá el siguiente proceso **para cada tipo de Kernel y para cada proporción de líneas**:

1. El conjunto de tipos de kernel se aplica a la matriz de datos *X*, asignando el valor numérico a la2 El argumento *arc_cosine_deep* y la proporción de líneas establecen el valor en el argumento de proporción de *filas*.
2. Luego realizamos cinco particiones aleatorias, con 80% los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función *cv_random*.
3. *Predictions*, *Times* e *Hyperparams* que se utilizarán para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función *gs_summaries*, además de guardar los tiempos de ejecución de cada modelo entrenado para cada partición.
4. **Para cada partición:**
 1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
 2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo valores entre 100 y 500 para el hiperparámetro *tree_number*,

valores entre 0,3 y 0,8 para el hiperparámetro *sampled_x_vars_number* y valores entre 5 y 10 para el hiperparámetro *node_size*, con "Optimización bayesiana" como tipo de tuneo (parámetro predeterminado de *tune_type*). ;

3. Con el modelo obtenido en (2), se predice la variable respuesta y en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
 4. Identificación de las predicciones del conjunto de prueba: Se crea el data frame *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba. Además, cada fila de *FoldPrediction* se agrega al data frame *Predictions**.
 5. Identificación del tiempo de ejecución del entrenamiento del modelo obtenido en (2): Se crea el data frame *FoldTime* que contiene la columna que especifica el tipo de *Kernel* utilizado para entrenar el modelo, el número de *Fold* y la columna *Minutos* que indica el tiempo de ejecución, en minutos, del entrenamiento del modelo obtenido en (2). Además, cada fila de *FoldTime* se agrega al data frame *Times*.
 6. Identificación de hiperparámetros; Se crea el data frame *HyperparamsFold* que contiene las columnas *tree_number*, *node_size*, *sampled_x_vars_number*, *mse* y *Fold*, donde *m* es el costo del modelo para cada combinación de los hiperparámetros especificados y el número de *Fold*. Además, cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
- *Predictions* contiene las columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo los hiperparámetros óptimos (entre las posibles combinaciones de estos) que minimizan la función de costo con el tipo de tuneo "Optimización bayesiana", correspondiente al formato necesario para utilizar la función *gs_summaries* en estas predicciones en el caso de variables continuas.
5. *Predictions* y con la ayuda de la función *gs_summaries*, evaluamos la capacidad predictiva del modelo entrenado para el tipo de kernel especificado. La función *gs_summaries* devuelve una lista que identificamos como *summaries* y que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.
 6. Finalmente, se crea un elemento con el nombre de kernel especificado en cada una de las listas *PredictionsAll*, *TimesAll*, *HyperparamsAll* y *SummariesAll*, que corresponden a los marcos de datos de las listas *Predictions*, *Times*, *Hyperparams* y *summaries*, respectivamente.

```
for (kernel in kernels) {  
  cat("*** Kernel:", kernel, "***\n")  
  for (line_proportion in lines_proportions) {  
    cat("\t*** Line_Proportion:", line_proportion, "***\n")  
  
    # Compute the kernel
```

```

X <- kernelize(
  X,
  kernel = kernel,
  arc_cosine_deep = 2,
  rows_proportion = line_proportion
)

# Random Partition
set.seed(2022)
folds <- cv_random(
  records_number = nrow(X),
  folds_number = 5,
  testing_proportion = 0.2
)

# Empty data frames that will contain Predictions, Times
# of execution & Summaries for each partition
Predictions <- data.frame()
Times <- data.frame()
Hyperparams <- data.frame()

for (i in seq_along(folds)) {
  cat("\t*** Fold:", i, "***\n")
  fold <- folds[[i]]

  # Identify the training and testing sets
  X_training <- X[fold$training, ]
  X_testing <- X[fold$testing, ]
  y_training <- y[fold$training]
  y_testing <- y[fold$testing]

  # Model training
  model <- random_forest(
    x = X_training,
    y = y_training,

    # Specify the hyperparameters values
    trees_number = list(min = 100, max = 500),
    sampled_x_vars_number = list(min = 0.3, max = 0.8),
    node_size = list(min = 5, max = 10),
    tune_cv_type = "random",
    tune_folds_number = 5,
    tune_bayes_samples_number = 5,
    tune_bayes_iterations_number = 5,
    tune_testing_proportion = 0.2,
    tune_type = "bayesian_optimization",
    tune_grid_proportion = 0.8,
    # In this example the iteration wont be shown
    verbose = FALSE
  )
}

```



```

)

# Testing Predictions
predictions <- predict(model, X_testing)

# Predictions for the Fold Fold
Predictions <- data.frame(
  Fold = i,
  Line = PhenoToy$Line[fold$testing],
  Env = PhenoToy$Env[fold$testing],
  Observed = y_testing,
  Predicted = predictions$predicted
)
Predictions <- rbind(Predictions, FoldPredictions)

# Execution times
FoldTime <- data.frame(
  kernel = kernel,
  Fold = i,
  Minutes = as.numeric(model$execution_time, units = "mins")
)
Times <- rbind(Times, FoldTime)

# Hyperparams for the Fold
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)
}

# Summaries of the Folds
summaries <- gs_summaries(Predictions)
str_line <- paste("Line_Proportion:", line_proportion)

# Predictions, Times of execution & Summaries for the
# specified Kernel
PredictionsAll[[kernel]][[str_line]] <- Predictions
TimesAll[[kernel]][[str_line]] <- Times
HyperparamsAll[[kernel]][[str_line]] <- Hyperparams
SummariesAll[[kernel]][[str_line]] <- summaries
}

#> *** Kernel: Sparse_Gaussian ***
#> *** Line_Proportion: 0.5 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.6 ***
#> *** Fold: 1 ***

```

```

#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.7 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.8 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.9 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 1 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: Sparse_Arc_cosine ***
#> *** Line_Proportion: 0.5 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.6 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.7 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.8 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***

```

```
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.9 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 1 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
```

Recordando que este proceso se realizó para cada combinación de tipo de kernel y relación de línea especificada, cada una de las listas *PredictionsAll*, *TimesAll*, *HyperparamsAll* y *SummariesAll* contiene las predicciones, los tiempos de ejecución, las combinaciones de hiperparámetros (en este caso *alpha*) los resúmenes, respectivamente, para cada combinación entre el tipo de kernel y la proporción de líneas aplicadas a la matriz de datos *X*. A modo de ejemplo, a continuación se muestran los resultados obtenidos para el tipo de kernel "Sparse_Gaussian" y "Line_Proportion: 0.9":

```
# Predictions for the Sparse_Gaussian Kernel
head(PredictionsAll$Sparse_Gaussian$`Line_Proportion: 0.9`)
#>   Fold      Line Env Observed Predicted
#> 1     5 CKDHL0129 KAK      5.18  5.641925
#> 2     5 CKDHL0647 KTI      3.85  5.820910
#> 3     5 CKDHL0054 KAK      5.37  6.234541
#> 4     5 CKDHL0647 KAK      3.08  5.804912
#> 5     5 CKDHL0052 EBU      5.88  5.837369
#> 6     5 CKDHL0437 KTI      7.17  6.224897
# Times of execution for the Sparse_Gaussian Kernel
TimesAll$Sparse_Gaussian$`Line_Proportion: 0.9`
#>           kernel Fold  Minutes
#> 1 Sparse_Gaussian    1 0.2660479
#> 2 Sparse_Gaussian    2 0.1370334
#> 3 Sparse_Gaussian    3 0.2811677
#> 4 Sparse_Gaussian    4 0.3002371
#> 5 Sparse_Gaussian    5 0.2762989
# Elements of SummariesAll
names(SummariesAll)
#> [1] "Sparse_Gaussian" "Sparse_Arc_cosine"
# Elements of summaries for Sparse_Gaussian Kernel
names(SummariesAll$Sparse_Gaussian)
#> [1] "Line_Proportion: 0.5" "Line_Proportion: 0.6"
#> [3] "Line_Proportion: 0.7" "Line_Proportion: 0.8"
#> [5] "Line_Proportion: 0.9" "Line_Proportion: 1"
names(SummariesAll$Sparse_Gaussian$`Line_Proportion: 0.9`)
```

```

#> [1] "line" "env" "fold"
# Summaries by Line
head(SummariesAll$Sparse_Gaussian$`Line_Proprtion: 0.9`$line)
#>      Line Observed Predicted Difference
#> 1 GID7628467    5.8605    5.8622    0.0017
#> 2 CKDHL0052    5.8800    5.8374    0.0426
#> 3 CKDHL0136    5.6600    5.7951    0.1351
#> 4 CKDHL0258    5.6150    5.7919    0.1769
#> 5 CKDHL0529    5.8100    6.1153    0.3053
#> 6 GID7730251    5.1354    4.8067    0.3287

# Summaries by Enviroment
SummariesAll$Sparse_Gaussian$`Line_Proprtion: 0.9`$env[, 1:8]
#>      Env      MSE MSE_SE  RMSE RMSE_SE  NRMSE NRMSE_SE
#> 1   EBU  0.1048      NA 0.3238      NA  1.4277      NA
#> 2   KAK  1.8583      NA 1.3632      NA  1.0558      NA
#> 3   KTI  0.7995      NA 0.8941      NA  0.8967      NA
#> 4  Bed5IR 0.9756      NA 0.9877      NA  1.9766      NA
#> 5   EHT  1.3857      NA 1.1772      NA  0.9181      NA
#> 6  Flat5IR 1.8476      NA 1.3593      NA  2.2849      NA
#> 7  FlatDrip 11.8721      NA 3.4456      NA 17.6764      NA
#> 8  Global  2.9702      NA 1.7234      NA  1.2148      NA
#>      MAE
#> 1 0.2806
#> 2 1.0770
#> 3 0.7152
#> 4 0.8665
#> 5 1.1129
#> 6 1.2841
#> 7 3.4264
#> 8 1.2763
SummariesAll$Sparse_Gaussian$`Line_Proprtion: 0.9`$env[, 9:15]
#>      MAE_SE      Cor Cor_SE Intercept Intercept_SE  Slope
#> 1   NA  0.7146      NA  -8.6748      NA  2.5303
#> 2   NA  0.2483      NA  -3.3553      NA  1.4391
#> 3   NA  0.3801      NA  -6.7557      NA  2.1096
#> 4   NA  0.4442      NA   3.6920      NA  0.4696
#> 5   NA  0.8393      NA  -6.6095      NA  2.4550
#> 6   NA  0.5716      NA   2.4570      NA  0.7733
#> 7   NA  0.4165      NA   1.5913      NA  0.1906
#> 8   NA -0.4149      NA  12.0339      NA -1.1483
#>      Slope_SE
#> 1   NA
#> 2   NA
#> 3   NA
#> 4   NA
#> 5   NA
#> 6   NA
#> 7   NA
#> 8   NA

```

```

SummariesAll$Sparse_Gaussian$`Line_Proprtion: 0.9`$env[, 16:19]
#>      R2 R2_SE  MAAPE MAAPE_SE
#> 1 0.5106   NA 0.0449      NA
#> 2 0.0617   NA 0.2392      NA
#> 3 0.1445   NA 0.1313      NA
#> 4 0.1974   NA 0.1359      NA
#> 5 0.7044   NA 0.1780      NA
#> 6 0.3267   NA 0.1933      NA
#> 7 0.1735   NA 0.8881      NA
#> 8 0.1722   NA 0.2709      NA

# Summaries by Fold
SummariesAll$Sparse_Gaussian$`Line_Proprtion: 0.9`$fold[, 1:8]
#>      Fold    MSE MSE_SE  RMSE RMSE_SE  NRMSE NRMSE_SE  MAE
#> 1      5 2.6919 1.5477 1.3644  0.372 3.7480  2.3302 1.2518
#> 2 Global 2.9702    NA 1.7234    NA 1.2148    NA 1.2763
SummariesAll$Sparse_Gaussian$`Line_Proprtion: 0.9`$fold[, 9:15]
#>      MAE_SE      Cor Cor_SE Intercept Intercept_SE  Slope
#> 1  0.383  0.5164 0.0775  -2.5221      1.9118  1.4239
#> 2    NA -0.4149    NA  12.0339      NA -1.1483
#>      Slope_SE
#> 1  0.3656
#> 2    NA
SummariesAll$Sparse_Gaussian$`Line_Proprtion: 0.9`$fold[, 16:19]
#>      R2 R2_SE  MAAPE MAAPE_SE
#> 1 0.3027 0.0867 0.2587  0.1074
#> 2 0.1722    NA 0.2709    NA

```

Además, los elementos e la lista *HyperparamsAll* contienen las columnas *tree_number*, *node_size*, *sampled_x_vars_number*, *mse* y *Fold*, donde el valor de la columna *mse* corresponde al costo del modelo para cada cobinación de los valores de partición e hiperparámetro, ordenados de menor a mayor . mayor dentro de cada partición.

```

# First rows of Hyperparams
head(HyperparamsAll$Sparse_Gaussian$`Line_Proprtion: 0.9`)
#>      trees_number node_size sampled_x_vars_number      mse Fold
#> 5              352         5              42 0.9510615      1
#> 7              326         8              25 0.9550356      1
#> 10             325         7              42 0.9572363      1
#> 3              367         5              61 0.9572677      1
#> 4              306         6              52 0.9573770      1
#> 2              442         8              41 0.9631317      1

# Last rows of Hyperparams
tail(HyperparamsAll$Sparse_Gaussian$`Line_Proprtion: 0.9`)
#>      trees_number node_size sampled_x_vars_number      mse Fold
#> 74              157        10              37 1.287963      5
#> 34              175         9              57 1.298549      5
#> 14              273         7              51 1.299458      5
#> 44              283         6              60 1.321589      5

```

#> 54	277	6	62 1.325300	5
#> 24	168	6	27 1.326718	5

6 Generalized boosted machines methods

6.1 Ejemplo con respuesta continua con Grid search y partciones aleatorias con solo G en el

predictor

Este ejemplo evalúa n modelo de máquina potenciada generalizada con cinco particiones aleatorias, con 20% los datos para el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de cada partición (los parámetros predeterminados de la función `cv_random`), para una respuesta continua, usando solo la matriz G (diseño de línea matriz que contiene la información genómica) como predictor y utilizando "Grid Search" como tipo de tuneo para los *hiperparámetros* `tree_number`, `node_size` y `encogimiento` (cabe mencionar que estos no son los únicos hiperparámetros ajustables del modelo).

En este ejemplo, el conjunto de datos utilizado es *EYTTToy* y busca predecir la variable continua *GY* del data frame *PhenoToy* utilizando la matriz G descrita anteriormente como predictor; por lo que identificamos las variables predictoras y de respuesta como X y y respectivamente.

```
# Load the data
load("EYTTToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy
# Data preparation of G
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %**% Geno

# Predictor and Response Variables
X <- LineG
y <- PhenoToy$GY

# Note that y is a continuous numeric vector
class(y)
#> [1] "numeric"
typeof(y)
#> [1] "double"
```

Tenga en cuenta que la variable de respuesta y es una variable continua (un vector con elementos de tipo "doble"), lo cual es importante para que el modelo se entrene automáticamente para este tipo de variable.

Posteriormente, realizamos cinco particiones aleatorias, con 80% los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función `cv_random` (con los parámetros por defecto). Además, creamos los data frames vacíos `Predictions` e `Hyperparams` que se utilizarán para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función `gs_summaries`.

```
# Set seed for reproducible results
set.seed(2022)
folds <- cv_random(records_number = nrow(X))

# A data frame that will contain the variables:
Predictions <- data.frame()
Hyperparams <- data.frame()
```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo los valores 30, 50 y 80 para el hiperparámetro *tree_number*, los valores 5, 15 y 20 para el hiperparámetro *node_size* y los valores 0.01, 0.01 y 0.1 para el hiperparámetro de contracción con "Grid Search" como tipo de tuneo (parámetro predeterminado de *tune_type*). Cabe señalar que estos no son los únicos hiperparámetros ajustables en el modelo;
3. Con el modelo obtenido en (2), se predice la variable respuesta *GY en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba.
 - b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*.
5. Identificación de hiperparámetros:
 - a. *HyperparamsFold* que contiene las columnas *tree_number*, *node_size*, *shrinkage*, *mse* y *Fold*, donde *mse* corresponde al costo del modelo para cada combinación de los hiperparámetros especificados y el número de *Fold*.
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
6. Se muestran los hiperparámetros óptimos del modelo obtenido en (2).

```
# Model training and predictions of the ith partition
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")
```

```

fold <- folds[[i]]

# Identify the training and testing sets
X_training <- X[fold$training, ]
X_testing <- X[fold$testing, ]
y_training <- y[fold$training]
y_testing <- y[fold$testing]

# Model training
model <- generalized_boosted_machine(
  x = X_training,
  y = y_training,

  # Specify the hyperparameters
  trees_number = c(30, 50, 80),
  node_size = c(5, 10, 15),
  max_depth = 5,
  shrinkage = c(0.001, 0.01, 0.1),
  tune_type = "grid_search",
  # In this example the iterations wont be shown
  verbose = FALSE
)

# Prediction of the test set
predictions <- predict(model, X_testing)

# Predictions for the Fold Fold
FoldPredictions <- data.frame(
  Fold = i,
  Line = PhenoToy$Line[fold$testing],
  Env = PhenoToy$Env[fold$testing],
  Observed = y_testing,
  Predicted = predictions$predicted
)
Predictions <- rbind(Predictions, FoldPredictions)

# Hyperparams for the Fold
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)

# Best hyperparams of the model
cat("*** Optimal hyperparameters: ***\n")
print(model$best_hyperparams)
}
#> *** Fold: 1 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 30

```



```
#>
#> $node_size
#> [1] 15
#>
#> $shrinkage
#> [1] 0.001
#>
#> $mse
#> [1] 3.174917
#>
#> *** Fold: 2 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 30
#>
#> $node_size
#> [1] 15
#>
#> $shrinkage
#> [1] 0.001
#>
#> $mse
#> [1] 3.097005
#>
#> *** Fold: 3 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 80
#>
#> $node_size
#> [1] 15
#>
#> $shrinkage
#> [1] 0.001
#>
#> $mse
#> [1] 3.161476
#>
#> *** Fold: 4 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 30
#>
#> $node_size
#> [1] 15
#>
#> $shrinkage
#> [1] 0.001
#>
#> $mse
```

```

#> [1] 2.874648
#>
#> *** Fold: 5 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 50
#>
#> $node_size
#> [1] 15
#>
#> $shrinkage
#> [1] 0.001
#>
#> $mse
#> [1] 2.972389

```

Predictions contiene las columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo los hiperparámetros óptimos (entre los valores posibles de las combinaciones de *tree_number*, *node_size* y *encogimiento*) que minimizan la función de coste con el tipo de tuneo "Grid Search", correspondiente al formato necesario para utilizar la función *gs_summaries* sobre estas predicciones en el caso de variables continuas.

La función *gs_summaries* devuelve una lista que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```

head(Predictions)
#>   Fold      Line      Env Observed Predicted
#> 1     1 GID7632666 FlatDrip 2.707868  5.462589
#> 2     1 GID7628158 Flat5IR  6.390845  5.448288
#> 3     1 GID7631195      EHT  7.424293  5.463721
#> 4     1 GID7628467 Flat5IR  7.193116  5.457200
#> 5     1 GID7630553 Flat5IR  7.794541  5.461766
#> 6     1 GID7629600 FlatDrip  2.379049  5.463735
unique(Predictions$Fold)
#> [1] 1 2 3 4 5
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted Difference
#> 1 GID7729805   5.5752   5.5908    0.0156
#> 2 GID7634730   5.5399   5.5661    0.0262
#> 3 GID7630551   5.5774   5.5392    0.0382
#> 4 GID7631604   5.4882   5.5525    0.0644
#> 5 GID7632527   5.6159   5.5482    0.0677
#> 6 GID7626381   5.7021   5.5641    0.1379

```

```

# Summaries by Environment
summaries$env[, 1:7]
#>      Env      MSE MSE_SE      RMSE RMSE_SE      NRMSE NRMSE_SE
#> 1  Bed5IR 1.0079 0.2837 0.9614 0.1446 1.9338 0.4703
#> 2    EHT 1.0744 0.1351 1.0270 0.0703 1.1619 0.0986
#> 3 Flat5IR 1.2916 0.1467 1.1295 0.0630 1.8911 0.1699
#> 4 FlatDrip 8.3208 0.2536 2.8832 0.0439 9.8867 1.4242
#> 5  Global 2.6310 0.2260 1.6157 0.0715 1.0107 0.0135
summaries$env[, 8:14]
#>      MAE MAE_SE      Cor Cor_SE Intercept Intercept_SE      Slope
#> 1 0.8366 0.1603 0.2216 0.1605 -89.2596 66.7068 17.4003
#> 2 0.8883 0.0576 0.4188 0.1573 -313.8345 150.0782 57.9766
#> 3 1.0233 0.0512 0.4473 0.1359 -133.5907 67.8363 25.3387
#> 4 2.8667 0.0393 0.3333 0.2286 -58.8295 71.5674 10.9089
#> 5 1.2990 0.0461 -0.3380 0.0884 260.4652 59.2131 -45.9873
summaries$env[, 15:19]
#>      Slope_SE      R2 R2_SE      MAAPE MAAPE_SE
#> 1 12.2361 0.1521 0.1010 0.1235 0.0217
#> 2 27.1801 0.2743 0.1554 0.1410 0.0124
#> 3 12.3423 0.2739 0.0811 0.1512 0.0068
#> 4 12.8867 0.3201 0.0729 0.8176 0.0107
#> 5 10.6829 0.1455 0.0659 0.2881 0.0215

# Summaries by Fold
summaries$fold[, 1:8]
#>      Fold      MSE MSE_SE      RMSE RMSE_SE      NRMSE NRMSE_SE      MAE
#> 1      1 3.0225 1.5199 1.6057 0.3848 3.6245 2.0668 1.4857
#> 2      2 3.0275 1.7659 1.5566 0.4489 4.1182 1.9569 1.4951
#> 3      3 2.6841 1.9562 1.3503 0.5357 3.6093 2.0363 1.2717
#> 4      4 3.1170 2.0032 1.5431 0.4953 2.5591 1.0348 1.4174
#> 5      5 2.7672 1.7872 1.4456 0.4752 4.6808 3.3125 1.3489
#> 6 Global 2.6310 0.2260 1.6157 0.0715 1.0107 0.0135 1.2990

```

Además, Hyperparams contiene las columns *tree_number*, *node_size*, *shrinkage*, *mse* y *Fold*, donde el valor de la columna *mse* corresponde al costo del modelo para cada combinación de hiperparámetros y partición, ordenados de menor a mayor dentro de cada partición.

```

# First rows of Hyperparams
head(Hyperparams)
#>      trees_number node_size shrinkage      mse Fold
#> 7              30         15    0.001 3.174917 1
#> 4              30         10    0.001 3.178474 1
#> 8              50         15    0.001 3.180376 1
#> 5              50         10    0.001 3.185529 1
#> 9              80         15    0.001 3.186439 1
#> 1             30          5    0.001 3.187583 1
# Last rows of Hyperparams
tail(Hyperparams)
#>      trees_number node_size shrinkage      mse Fold

```

```
#> 224      30      10      0.1 3.880113      5
#> 234      50      10      0.1 4.175070      5
#> 194      30       5      0.1 4.394469      5
#> 244      80      10      0.1 4.523936      5
#> 204      50       5      0.1 5.189609      5
#> 215      80       5      0.1 5.195949      5
```

6.2 Ejemplo con respuesta binaria con Grid search y validación cruzada 7-fold con Env+G en el predictor

Este ejemplo evalúa un modelo de máquina potenciada generalizada con validación cruzada 7-fold, para una respuesta binaria, utilizando el efecto de ambiente y la matriz G como predictores, además de "búsqueda de cuadrícula" como tipo de tuneo para los hiperparámetros *tree_number*, *node_size* y *cogimiento*.

En este ejemplo, el conjunto de datos utilizado es *GroundnutToy* y el objetivo es predecir la variable binaria y_{bin} , que es una transformación de la variable YPH de *PhenoToy* utilizando la función *ntile*, indicando si la respuesta es mayor que la mediana de esta variable o no. es, utilizando la matriz de diseño de la variable *PhenoToy* Env y la matriz, G descrita anteriormente, como predictores; por lo que identificamos las variables predictoras y de respuesta como X y y_{bin} respectivamente.

```
# Load the data
load("GroundnutToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of G
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %*% Geno

# Predictor and Response Variables
X <- cbind(Env, LineG)
y_bin <- BurStMisc::ntile(PhenoToy$YPH, 2, result = "factor")
y_bin <- factor(y_bin, ordered = FALSE)
```

Tenga en cuenta que la variable de respuesta y_{bin} es un factor con solo dos niveles (o categorías), lo cual es importante para que el modelo se entrene automáticamente para una variable binaria (regresión logística). Por esta razón, es importante tener en cuenta esas variables de respuesta binarias o categóricas antes de usar la función *generalized_boosted_machine*.

Posteriormente hacemos las particiones correspondientes a 7-Fold CV, con la ayuda de la función *cv*. Además, creamos los data frames vacíos *Predictions* e *Hyperparams* que se

utilizarán para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función `gs_summaries*`.

```
# Set seed for reproducible results
set.seed(2022)
folds <- cv_kfold(records_number = nrow(X), k = 7)

# A data frame that will contain the variables:
Predictions <- data.frame()
Hyperparams <- data.frame()
```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
2. El modelo se entra con el conjunto de entrenamiento. Esto se hace proponiendo los valores 30, 50 y 80 para el hiperparámetro *tree_number*, los valores 5, 10 y 15 para el hiperparámetro *node_size* y los valores 0.001, 0.01 y 0.1 para el hiperparámetro de contracción, con "Grid Search" como tipo de tuneo. (parámetro predeterminado de *tune_type*). Cabe señalar que estos no son los únicos hiperparámetros ajustables en el modelo;
3. Con el modelo obtenido en (2), y_{bin} se predice la variable respuesta en el conjunto de prueba, con el fin de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: *Fold number*, *Line*, *Env*, *Observed*, *Predicted*, 1 y 2 para cada elemento del conjunto de prueba. Tenga en cuenta que, a diferencia del ejemplo anterior, ahora tenemos dos columnas adicionales correspondientes a las probabilidades asociadas con cada elemento correspondiente a esa categoría.
 - b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*. Con esto, *Predictions* se formatea para que sea un argumento de la función *gs_summaries*.
5. Identificación de hiperparámetros:
 - a. *HyperparamsFold* que contiene las columnas *tree_number*, *node_size*, *accuracy* y *Fold*, donde *accuracy* es la precisión del modelo para cada combinación de los hiperparámetros especificados y el número de *Fold*.
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
6. Se muestran los hiperparámetros óptimos.

```

# Model training and predictions of the ith partition
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")
  fold <- folds[[i]]

  # Identify the training and testing sets
  X_training <- X[fold$training, ]
  X_testing <- X[fold$testing, ]
  y_training <- y_bin[fold$training]
  y_testing <- y_bin[fold$testing]

  # Model training
  model <- generalized_boosted_machine(
    x = X_training,
    y = y_training,

    # Specify the hyperparameters
    trees_number = c(30, 50, 80),
    node_size = c(5, 10, 15),
    max_depth = 5,
    shrinkage = c(0.001, 0.01, 0.1),
    sampled_records_proportion = 0.6,
    tune_type = "grid_search",
    # In this example the iterations wont be shown
    verbose = FALSE
  )

  # Prediction of the test set
  predictions <- predict(model, X_testing)

  # Predictions for the Fold Fold
  FoldPredictions <- cbind(
    data.frame(
      Fold = i,
      Line = PhenoToy$Line[fold$testing],
      Env = PhenoToy$Env[fold$testing],
      Observed = y_testing,
      Predicted = predictions$predicted
    ),
    predictions$probabilities
  )

  Predictions <- rbind(Predictions, FoldPredictions)

  # Hyperparams
  HyperparamsFold <- model$hyperparams_grid %>%
    mutate(Fold = i)
  Hyperparams <- rbind(Hyperparams, HyperparamsFold)

```

```

# Best hyperparams of the model
cat("*** Optimal hyperparameters: ***\n")
print(model$best_hyperparams)
}
#> *** Fold: 1 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 80
#>
#> $node_size
#> [1] 15
#>
#> $shrinkage
#> [1] 0.01
#>
#> $accuracy
#> [1] 0.6380952
#>
#> *** Fold: 2 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 50
#>
#> $node_size
#> [1] 15
#>
#> $shrinkage
#> [1] 0.01
#>
#> $accuracy
#> [1] 0.58
#>
#> *** Fold: 3 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 30
#>
#> $node_size
#> [1] 15
#>
#> $shrinkage
#> [1] 0.1
#>
#> $accuracy
#> [1] 0.5947619
#>
#> *** Fold: 4 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 80

```

```
#>
#> $node_size
#> [1] 15
#>
#> $shrinkage
#> [1] 0.01
#>
#> $accuracy
#> [1] 0.5833333
#>
#> *** Fold: 5 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 30
#>
#> $node_size
#> [1] 15
#>
#> $shrinkage
#> [1] 0.1
#>
#> $accuracy
#> [1] 0.6033333
#>
#> *** Fold: 6 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 30
#>
#> $node_size
#> [1] 15
#>
#> $shrinkage
#> [1] 0.1
#>
#> $accuracy
#> [1] 0.6314286
#>
#> *** Fold: 7 ***
#> *** Optimal hyperparameters: ***
#> $trees_number
#> [1] 30
#>
#> $node_size
#> [1] 15
#>
#> $shrinkage
#> [1] 0.01
#>
```



```
#> $accuracy
#> [1] 0.5609524
```

Predictions contiene as columnas *Fold*, *Line*, *Env*, *Observed*, *Predicted*, 1 y 2 para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo los hiperparámetros óptimos (entre los valores posibles de las combinaciones de *tree_number*, *node_size* y *shrinkage*) que minimizan la función de costo con el tipo de tuneo "Grid Search", correspondiente al formato necesario para usar la función *gs_summaries* en estas predicciones e el caso de variables binarias.

La función *gs_summaries* devuelve una lista que contiene tres marcos de datos correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```
head(Predictions[, 2:7])
#>           Line           Env Observed Predicted      1
#> 1 49x37-99(b)taLL  ICRISAT_R15         1         1 0.5960387
#> 2   CSMG84-1    JALGOAN_R15         2         1 0.6114181
#> 3   DTG15  ALIYARNAGAR_R15         1         1 0.5753067
#> 4   DTG3  ICRISAT_PR15-16         2         1 0.5045079
#> 5   Gangapuri    JALGOAN_R15         1         1 0.6002104
#> 6   ICG15419    JALGOAN_R15         2         1 0.6683217
#>           2
#> 1 0.4039613
#> 2 0.3885819
#> 3 0.4246933
#> 4 0.4954921
#> 5 0.3997896
#> 6 0.3316783
unique(Predictions$Fold)
#> [1] 1 2 3 4 5 6 7

# Summaries
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summaries$line)
#>           Line Observed Predicted      X1      X2
#> 1   49x37-134         1         1 0.5889 0.4111
#> 2 49x37-99(b)taLL         1         1 0.5935 0.4065
#> 3   CSMG84-1         1         1 0.5367 0.4633
#> 4   DTG15         1         1 0.5490 0.4510
#> 5   DTG3         2         1 0.5118 0.4882
#> 6   Gangapuri         1         1 0.6243 0.3757

# Summaries by Environment
summaries$env[, 1:5]
#>           Env      PCCC PCCC_SE      Kappa Kappa_SE
```

```

#> 1 ALIYARNAGAR_R15 0.7071 0.0638 0.4177 0.1220
#> 2 ICRISAT_PR15-16 0.6184 0.1061 -0.0505 0.2010
#> 3 ICRISAT_R15 0.8031 0.0794 0.6108 0.1576
#> 4 JALGOAN_R15 0.5000 0.1161 0.0419 0.2077
#> 5 Global 0.6275 0.0411 0.2395 0.0799
summaries$env[, 6:7]
#> BrierScore BrierScore_SE
#> 1 0.4593 0.0538
#> 2 0.4975 0.0559
#> 3 0.4018 0.0388
#> 4 0.5249 0.0707
#> 5 0.4845 0.0184

# Summaries by Fold
summaries$fold
#> Fold PCCC PCCC_SE Kappa Kappa_SE BrierScore
#> 1 1 0.6804 0.1283 0.3241 0.2869 0.4750
#> 2 2 0.5250 0.2056 0.0881 0.3561 0.4647
#> 3 3 0.6280 0.0693 0.2568 0.1781 0.4693
#> 4 4 0.7571 0.1445 0.3333 0.3005 0.4180
#> 5 5 0.6667 0.1179 0.2500 0.2500 0.4724
#> 6 6 0.6167 0.1686 0.1955 0.3637 0.5347
#> 7 7 0.7262 0.1160 0.4329 0.2241 0.4621
#> 8 Global 0.6275 0.0411 0.2395 0.0799 0.4845
#> BrierScore_SE
#> 1 0.0721
#> 2 0.0673
#> 3 0.0636
#> 4 0.0559
#> 5 0.1065
#> 6 0.1332
#> 7 0.0212
#> 8 0.0184

```

Además, Hyperparams contiene las columnas *tree_number*, *node_size*, *shrinkage*, *preision* y *Fold*, donde el valor de la columna de precisión corresponde a la precisión del modelo para cada combinación de hiperparámetro y valores de partición, ordenados de menor a mayor dentro de cada partición.

```

# First rows of Hyperparams
head(Hyperparams)
#> trees_number node_size shrinkage accuracy Fold
#> 18 80 15 0.01 0.6380952 1
#> 25 30 15 0.10 0.6280952 1
#> 17 50 15 0.01 0.6276190 1
#> 16 30 15 0.01 0.6076190 1
#> 15 80 10 0.01 0.6076190 1
#> 26 50 15 0.10 0.5980952 1

# Last rows of Hyperparams
tail(Hyperparams)

```

```
#>      trees_number node_size shrinkage accuracy Fold
#> 226           30      10      0.100 0.4738095    7
#> 56            50      10      0.001 0.4657143    7
#> 120           30       5      0.001 0.4561905    7
#> 86            50      15      0.001 0.4466667    7
#> 76            30      15      0.001 0.4466667    7
#> 46            30      10      0.001 0.4466667    7
```

6.3 Ejemplo con variable respuesta categórica con optimización bayesiana con línea de partición aleatoria *Env* + *G* + *GE* dentro del predictor

Este ejemplo evalúa un modelo de generalized boosted machine con cinco particiones aleatorias del conjunto de líneas, con 20% las líneas para el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de cada partición (los parámetros predeterminados de la función `cv_random`), para una respuesta categórica, utilizando el efecto del Medio Ambiente, la matriz *G* y la interacción entre estos dos como predictores, además e utilizar la "Optimización Bayesiana" como un tipo de tuneo de hiperparámetros.

En este ejemplo, el conjunto de datos utilizado es *ChickpeaToy* y buscamos predecir la variable categórica *y*, que es una transformación de la variable *Biomass del marco de datos PhenoToy* usando la función *ntile*, usando la matriz de diseño de la variable *Env* de *PhenoToy*, la matriz *G* descrito anteriormente y la matriz de diseño de la interacción entre estos dos, como predictores; por lo que identificamos las variables predictoras y de respuesta como *X* y *y* respectivamente.

```
# Load the dataset
load("ChickpeaToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env, G & GE
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %*% Geno
LineGenoxEnv <- model.matrix(~ 0 + LineG:Env)

# Predictor and Response Variables
X <- cbind(Env, LineG, LineGenoxEnv)
y <- BurStMisc::ntile(PhenoToy$Biomass, 3, result = "factor")
y <- factor(y, ordered = FALSE)

# First 30 responses
print(y[1:30])
#> [1] 2 3 2 3 3 1 1 2 1 2 3 1 2 3 2 2 3 2 3 3 3 2 3 1 2 3 2 2 3
```

```
#> [30] 2  
#> Levels: 1 2 3
```

Tenga en cuenta que la variable de respuesta *y* es un factor con tres niveles (o categorías), lo cual es importante para que el modelo se entrene automáticamente para una variable categórica (*modelo multinomial simétrico*). Por esta razón, es importante tener en cuenta esas variables de respuesta binarias o categóricas antes de usar la función *generalized_boosted_machine*.

Posteriormente, realizamos cinco particiones aleatorias del conjunto de líneas, con 80% este conjunto para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función *cv_random* (con los parámetros por defecto). Además, creamos los data frames vacíos *Predictions* e *Hyperparams* que servirán para guardar los valores observados y predichos en cada ambiente y posteriormente evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
# Set seed for reproducible results  
set.seed(2022)  
# Unique Lines  
GIDs <- unique(PhenoToy$Line)  
folds <- cv_random(length(GIDs))  
  
# A data frame that will contain the variables:  
Predictions <- data.frame()  
Hyperparams <- data.frame()
```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta, identificando primero las líneas correspondientes a este conjunto;
2. El modelo se entrena con el conjunto de entrenamiento. Eso se hace proponiendo valores entre 10 y 100 para el hiperparámetro *tree_number*, valores entre 5 y 15 para el hiperparámetro *node_size* y valores entre 0.001 y 0.1 para el hiperparámetro de contracción, con "Optimización bayesiana" como tipo de tuneo. Cabe señalar que estos no son los únicos hiperparámetros ajustables en el modelo;
3. Con el modelo obtenido en (2), se predice la variable respuesta *y* en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed*, *Predicted*, 1, 2 y 3 para cada elemento del conjunto de prueba. Tenga en cuenta que, a diferencia de los ejemplos anteriores, ahora tenemos tres columnas adicionales correspondientes a las probabilidades asociadas con cada elemento correspondiente a esa categoría.

- b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*. Con esto, *Predictions* se formatea para que sea un argumento de la función *gs_summaries*.
- 5. Identificación de hiperparámetros:
 - a. *HyperparamsFold* que contiene las columnas *número_árboles*, *tamaño_nodo*, *contracción*, *precisión* y *Plegado*, donde *precisión* es la precisión del modelo para cada combinación de los hiperparámetros especificados y el número de pliegue.
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
- 6. Se muestran los hiperparámetros óptimos.

```
# Model training and predictions of the ith partition
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")

  # Identify the training and testing Line sets
  fold <- folds[[i]]
  Lines_sam_i <- GIDs[fold$training]
  fold_i <- which(PhenoToy$Line %in% Lines_sam_i)

  # Identify the training and testing sets
  X_training <- X[fold_i, ]
  X_testing <- X[-fold_i, ]
  y_training <- y[fold_i]
  y_testing <- y[-fold_i]

  # Model training
  model <- generalized_boosted_machine(
    x = X_training,
    y = y_training,

    # Specify the hyperparameters
    trees_number = list(min = 10, max = 100),
    node_size = list(min = 5, max = 15),
    max_depth = 5,
    shrinkage = list(min = 0.001, max = 0.1),
    tune_type = "Bayesian_optimization",
    tune_bayes_samples_number = 5,
    tune_bayes_iterations_number = 5,

    # In this example the iterations wont bw shown
    verbose = FALSE
  )

  # Prediction of the test set
```

```

predictions <- predict(model, X_testing)

# Predictions for the Fold Fold
FoldPredictions <- cbind(
  data.frame(
    Fold = i,
    Line = PhenoToy$Line[-fold_i],
    Env = PhenoToy$Env[-fold_i],
    Observed = y_testing,
    Predicted = predictions$predicted
  ),
  predictions$probabilities
)

Predictions <- rbind(Predictions, FoldPredictions)

# Hyperparams
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)

# Best hyperparams of the model
cat("*** Optimal hyperparameters: ***\n")
print(model$best_hyperparams)
}
#> *** Fold: 1 ***
#> $trees_number
#> [1] 98
#>
#> $node_size
#> [1] 5
#>
#> $shrinkage
#> [1] 0.001
#>
#> $accuracy
#> [1] 0.7014778
#>
#> *** Fold: 2 ***
#> $trees_number
#> [1] 100
#>
#> $node_size
#> [1] 5
#>
#> $shrinkage
#> [1] 0.1
#>
#> $accuracy

```

```

#> [1] 0.6534483
#>
#> *** Fold: 3 ***
#> $trees_number
#> [1] 94
#>
#> $node_size
#> [1] 9
#>
#> $shrinkage
#> [1] 0.005103461
#>
#> $accuracy
#> [1] 0.63867
#>
#> *** Fold: 4 ***
#> $trees_number
#> [1] 100
#>
#> $node_size
#> [1] 8
#>
#> $shrinkage
#> [1] 0.002423798
#>
#> $accuracy
#> [1] 0.6665025
#>
#> *** Fold: 5 ***
#> $trees_number
#> [1] 58
#>
#> $node_size
#> [1] 5
#>
#> $shrinkage
#> [1] 0.001
#>
#> $accuracy
#> [1] 0.6807882

```

Predictions contiene las columnas *Fold*, *Line*, *Env*, *Observed*, *Predicted*, 1, 2 y 3 para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo los hiperparámetros óptimos (entre los valores posibles de las combinaciones de *trees_nuber* y *node_size*) que minimizan la función de costo (*pcic*: Proporción de Casos Incorrectamente Clasificados) con el tipo de tuneo "Optimización Bayesiana", correspondiente al formato necesario para usar la función *gs_summaries en Predicción* en el caso de variables categóricas.

La función `gs_summaries` devuelve una lista que contiene tres marcos de dato correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```
head(Predictions)
#>   Fold      Line Env Observed Predicted      1      2
#> 1 1 ICCV03104 1      3      2 0.3130980 0.3664061
#> 2 1 ICCV03104 2      3      3 0.2846053 0.3260582
#> 3 1 ICCV03104 4      3      2 0.3057480 0.3807822
#> 4 1 ICCV03104 5      2      2 0.3070712 0.3810587
#> 5 1 ICCV03104 6      3      3 0.2805461 0.3186691
#> 6 1 ICCV03104 7      1      2 0.3398150 0.3482495
#>      3
#> 1 0.3204959
#> 2 0.3893365
#> 3 0.3134697
#> 4 0.3118701
#> 5 0.4007849
#> 6 0.3119354
unique(Predictions$Fold)
#> [1] 1 2 3 4 5
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted      X1      X2      X3
#> 1 ICCV00402      3      3 0.2845 0.3371 0.3784
#> 2 ICCV01301      1      3 0.2927 0.3359 0.3714
#> 3 ICCV03104      3      2 0.3051 0.3535 0.3413
#> 4 ICCV03105      2      3 0.2925 0.3301 0.3774
#> 5 ICCV03107      3      2 0.2474 0.4046 0.3479
#> 6 ICCV03109      2      2 0.3013 0.3339 0.3649

# Summaries by Environment
summaries$env
#>      Env      PCCC PCCC_SE      Kappa Kappa_SE BrierScore
#> 1 1 0.3333 0.0745 0.0935 0.0575 0.7731
#> 2 2 0.7000 0.0624 -0.0400 0.0400 0.6014
#> 3 4 0.5333 0.0624 0.2604 0.0642 0.6546
#> 4 5 0.5000 0.0913 -0.0400 0.0400 0.7792
#> 5 6 0.8333 0.0527 0.0000 0.0000 0.4504
#> 6 7 0.8667 0.0624 -0.0667 0.0516 0.4119
#> 7 Global 0.3667 0.0624 0.0943 0.0422 0.6963
#>      BrierScore_SE
#> 1 0.1193
#> 2 0.0133
#> 3 0.0457
#> 4 0.1602
```



```

#> 5      0.0980
#> 6      0.1043
#> 7      0.0329

# Summaries by Fold
summaries$fold
#>      Fold  PCCC PCCC_SE  Kappa Kappa_SE BrierScore
#> 1      1 0.6389 0.0512 0.0500 0.0707 0.6135
#> 2      2 0.5833 0.1537 -0.0375 0.0872 0.6988
#> 3      3 0.6389 0.0512 0.0918 0.0600 0.5344
#> 4      4 0.5833 0.1198 0.0067 0.0067 0.5852
#> 5      5 0.6944 0.1002 0.0857 0.0782 0.6269
#> 6 Global 0.3667 0.0624 0.0943 0.0422 0.6963
#>      BrierScore_SE
#> 1      0.0132
#> 2      0.2370
#> 3      0.0441
#> 4      0.0279
#> 5      0.0118
#> 6      0.0329

```

Además, Hyperparas contiene las columnas *tree_number*, *node_size*, *shrinkage*, *precisio* y *Fold*, donde el valor de la columna de precisión corresponde a la precisión del modelo para cada combinación de hiperparámetro y valores de partición, ordenados de menor a mayor dentro de cada partición.

```

# First rows of Hyperparams
head(Hyperparams)
#>   trees_number node_size shrinkage accuracy Fold
#> 6           98         5 0.001000000 0.7014778 1
#> 9          100         6 0.001000000 0.6945813 1
#> 7          100         5 0.100000000 0.6736453 1
#> 3           77         8 0.004895483 0.6669951 1
#> 1           44         6 0.038224968 0.6665025 1
#> 2           66         8 0.033703027 0.6598522 1

# Last rows of Hyperparams
tail(Hyperparams)
#>   trees_number node_size shrinkage accuracy Fold
#> 84           49         6 0.09848852 0.6320197 5
#> 14           77         9 0.08937011 0.6187192 5
#> 44           40        10 0.09376227 0.6184729 5
#> 34           23        12 0.09494527 0.5842365 5
#> 24           29        13 0.07735142 0.4453202 5
#> 54           66        14 0.09203007 0.4381773 5

```

6.4 Ejemplo con kernels con Grid search y particiones aleatorias

Este ejemplo evalúa un modelo de Máquina Impulsada Generalizada con cinco particiones aleatorias, con 20% los datos para el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de cada partición (los parámetros predeterminados de la función

cv_random), para una respuesta continua, usndo la matriz de diseño de la variable PhenoToy Env, la matri descrita G anteriormente y la matriz de diseño de la interacción entre estas dos, coo predictores; así como el uso de "Grid Search" como u tipo de tuneo para los hiperparámetros *trees_number*, *ode_size* y *encogimiento*. Todo esto para los tipos de Kernel: "Linear", "Polnomio", "Sigmoid", "Gaussian", "Exponential", "Arc_cosine" y "Arc_cosine_L".

En este ejemplo, el conjunto de datos utilizado es *GroudnutToy* y el objetivo es predecir la variable continua *YPH* del data frame *PhenoToy* utilizando la matriz de diseño de la variable PhenoToy Env, la matriz descrita G anteriormente y la matriz de diseño de la interacción entre estos dos, como predictores; por lo que identificamos las variables predictoras y de respuesta como X y y respectivamente.

```
# Load the data
load("GroundnutToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env, G & GE
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LinexGeno <- Line %*% Geno
LinexGenoxEnv <- model.matrix(~ 0 + LinexGeno:Env)

# Predictor and Response Variables
X <- cbind(Env, LinexGeno, LinexGenoxEnv)
y <- PhenoToy$YPH

dim(X)
#> [1] 120 154
print(y[1:7])
#> [1] 746.90 1614.19 1454.29 998.40 754.60 735.82 1034.72
typeof(y)
#> [1] "double"
```

Tenga en cuenta que la variable e respuesta y es una variable continua (un vector con elementos de ipo "doble"), lo cual es importante par que el modelo se entrene automáticamente para una variable continua.

A diferencia de los ejemplos anteriores, ahora buscamos evaluar el modelo para cada tipo de kernel mencionado anteriormente. Por ello, creamos un vector en el que indicamos los tipo de kernel que queremos aplicar a la matriz X . Además, creamos las listas vacías *PredictionsAll*, *TimesAll*, *HyperparamsAll** y *SummariesAll* que se utilizarán ara guardar las predicciones, los tiempos de ejecución, los hiperparámetros y los esúmenes de cada modelo entrenado, es decir, para cada tipo de krnel; lo que a su vez servirá para guardar los valores

observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
kernels <- c(
  "linear",
  "polynomial",
  "sigmoid",
  "Gaussian",
  "exponential",
  "arc_cosine",
  "Arc_cosine_L"
)

# Empty lists that will contain Predictions,
# Times of execution & Summaries for each type of kernel
PredictionsAll <- list()
TimesAll <- list()
HyperparamsAll <- list()
SummariesAll <- list()
```

Posteriormente, se seguirá el siguiente proceso **para cada tipo de Kernel**:

1. identifique la variable *arc_deep* con el valor 2. Si el tipo de Kernel es "Arc_osine_L", el valor de la variable *arc_deep* se cambia a 3 y el *kernel_type* se identifica como "Arc_cosine"; de lo contrario, *kernel_type* se identifica como el kernel predeterminado.
2. El tipo de kernel establecido en (1) se aplica a la matriz de datos *X*, asignando al argumento *arc_cosine_deep* el valor establecido en la variable *arc_deep*. Tenga en cuenta que el argumento *arc_cosine_deep* se ignora si el tipo de kernel no es *Arc_cosine*.
3. Luego realizamos cinco particiones aleatorias, con 80% los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función *cv_random*.
4. Predictions, *Times* e Hyperparams que se utilizarán para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función *gs_summaries*, además de guardar los tiempos de ejecución de cada modelo entrenado para cada partición.
5. **Para cada partición:**
 1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
 2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo los valores 100, 300 y 500 para el hiperparámetro *tree_number*, los valores 5, 10 y 15 para el hiperparámetro *node_size* y los valores 0.001, 0.01 y 0.1 para el hiperparámetro de contracción, con "Grid Search" como tipo de ajuste . (parámetro predeterminado de *tune_type*);

3. Con el modelo obtenido en (2), se predice la variable respuesta y en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
 4. Identificación de las predicciones del conjunto de prueba: Se crea el data frame *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba. Además, cada fila de *FoldPrediction* se agrega al data frame *Predictions**.
 5. Identificación del tiempo de ejecución del entrenamiento del modelo obtenido en (2): Se crea el data frame *FoldTime* que contiene la columna que especifica el tipo de *Kernel* utilizado para entrenar el modelo, el número de *Fold* y la columna *Minutos que indica el* tiempo de ejecución, en minutos, del entrenamiento del modelo obtenido en (2). Además, cada fila de *FoldTime* se agrega al data frame *Times*.
 6. Identificación de hiperparámetros; Se crea el data frame *HyperparamsFold* que contiene las columnas *tree_number*, *node_size*, *shrinkage*, *mse* y *Fold*, donde *mse* corresponde al costo del modelo para cada combinación de los hiperparámetros especificados y el número de *Fold*. Además, cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
- data frame de predicciones contiene columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba para cada partición, donde las predicciones se realizan eligiendo los hiperparámetros óptimos (entre los posibles valores especificados) que minimizan la función de costo con el " Grid Search", correspondiente al formato necesario para utilizar la función *gs_summaries* sobre estas predicciones en el caso de variables continuas.
6. *Predictions* y con la ayuda de la función *gs_summaries*, evaluamos la capacidad predictiva del modelo entrenado para el tipo de kernel especificado. La función *gs_summaries* devuelve una lista que identificamos como *summaries* y que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.
 7. Finalmente, se crea un elemento con el nombre de kernel especificado en cada una de las listas *PredictionsAll*, *TimesAll*, *HyperparamsAll* y *SummariesAll*, que corresponden a los marcos de datos de las listas *Predictions*, *Times*, *Hyperparams* y *summaries*, respectivamente.

```
for (kernel in kernels) {
  cat("*** Kernel:", kernel, "***\n")

  # Identify the arc_deep and the kernel
  arc_deep <- 2
  if (kernel == "Arc_cosine_L") {
    arc_deep <- 3
    kernel <- "arc_cosine"
  } else {
    kernel <- kernel
  }
}
```

```

}

# Compute the kernel
X <- kernelize(X, kernel = kernel, arc_cosine_deep = arc_deep)

# Random Partition
set.seed(2022)
folds <- cv_random(
  records_number = nrow(X),
  folds_number = 5,
  testing_proportion = 0.2
)

# Empty data frames that will contain Predictions, Times
# of execution & Summaries for each partition
Predictions <- data.frame()
Times <- data.frame()
Hyperparams <- data.frame()

for (i in seq_along(folds)) {
  cat("\t*** Fold:", i, "***\n")
  fold <- folds[[i]]

  # Identify the training and testing sets
  X_training <- X[fold$training, ]
  X_testing <- X[fold$testing, ]
  y_training <- y[fold$training]
  y_testing <- y[fold$testing]

  # Model training
  model <- generalized_boosted_machine(
    x = X_training,
    y = y_training,

    # Specify the hyperparameters values
    trees_number = c(100, 300, 500),
    node_size = c(5, 10, 15),
    max_depth = 5,
    shrinkage = c(0.001, 0.01, 0.1),
    sampled_records_proportion = 0.6,
    tune_type = "grid_search",
    tune_grid_proportion = 0.8,
    # In this example the iterations wont be shown
    verbose = FALSE
  )

  # Testing Predictions
  predictions <- predict(model, X_testing)

```

```

# Predictions for the Fold Fold
FoldPredictions <- data.frame(
  Fold = i,
  Line = PhenoToy$Line[fold$testing],
  Env = PhenoToy$Env[fold$testing],
  Observed = y_testing,
  Predicted = predictions$predicted
)
Predictions <- rbind(Predictions, FoldPredictions)

# Execution times
FoldTime <- data.frame(
  kernel = kernel,
  Fold = i,
  Minutes = as.numeric(model$execution_time, units = "mins")
)
Times <- rbind(Times, FoldTime)

# Hyperparams for the Fold
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)
}

# Summaries of the Folds
summaries <- gs_summaries(Predictions)

# Predictions, Times of execution & Summaries for the
# specified Kernel
PredictionsAll[[kernel]] <- Predictions
TimesAll[[kernel]] <- Times
HyperparamsAll[[kernel]] <- Hyperparams
SummariesAll[[kernel]] <- summaries
}

#> *** Kernel: linear ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: polynomial ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: sigmoid ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***

```

```

#> *** Fold: 5 ***
#> *** Kernel: Gaussian ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: exponential ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: arc_cosine ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: Arc_cosine_L ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***

```

Recordando que este proceso se realizó para cada tipo de kernel, cada una de las listas *PredictionsAll*, *TimesAll*, *HyperparamsAll* y *SummariesAll* contiene las predicciones, los tiempos de ejecución, las combinaciones de los hiperparámetros con su correspondiente costo y los resúmenes, respectivamente, para cada tipo de kernel aplicado a la matriz de datos X. A modo de ejemplo, a continuación se muestran los resultados obtenidos para el tipo de kernel “Gaussian”:

```

# Predictions for the Gaussian Kernel
head(PredictionsAll$Gaussian)
#>   Fold      Line      Env Observed Predicted
#> 1     1 ICGV97115  JALGOAN_R15   817.85  1959.942
#> 2     1  ICG9315   ICRISAT_R15  1324.07  1194.479
#> 3     1 ICGV06099 ICRISAT_PR15-16 2334.15  2349.431
#> 4     1 ICGV00248   ICRISAT_R15  1993.36  1915.336
#> 5     1 ICGV05057   ICRISAT_R15  1856.64  2015.022
#> 6     1 ICGV02434  JALGOAN_R15   367.32  2019.677
# Times of execution for the Gaussian Kernel
TimesAll$Gaussian
#>   kernel Fold  Minutes
#> 1 Gaussian    1 0.2146087
#> 2 Gaussian    2 0.2216479
#> 3 Gaussian    3 0.2198558
#> 4 Gaussian    4 0.2566729
#> 5 Gaussian    5 0.2285512

```

```

# Elements of SummariesAll
names(SummariesAll)
#> [1] "linear"      "polynomial"  "sigmoid"     "Gaussian"
#> [5] "exponential" "arc_cosine"
# Elements of summaries for the Gaussian Kernel
names(SummariesAll$Gaussian)
#> [1] "line" "env"  "fold"
# Summaries by Line
head(SummariesAll$Gaussian$line)
#>      Line Observed Predicted Difference
#> 1 ICGV00248 2029.993  2053.720    23.7263
#> 2      DTG15 1696.928  1666.396    30.5320
#> 3 Gangapuri 1232.337  1185.876    46.4610
#> 4   ICG9315 1453.340  1514.130    60.7905
#> 5 49x37-134 1100.595  1166.437    65.8419
#> 6 ICGV05057 1962.590  1883.851    78.7394

# Summaries by Environment
SummariesAll$Gaussian$env[, 1:7]
#>      Env      MSE    MSE_SE      RMSE  RMSE_SE
#> 1 ALIYARNAGAR_R15 232929.4  67895.79  462.1452  69.5543
#> 2 ICRISAT_PR15-16 443810.1 157905.43  632.3545 104.8068
#> 3   ICRISAT_R15 222589.8  70941.53  450.9312  69.3738
#> 4   JALGOAN_R15 1074754.0 116052.88 1030.9011  54.7651
#> 5      Global 516453.5  83832.73  710.1459  55.1052
#>      NRMSE NRMSE_SE
#> 1 1.3906  0.7790
#> 2 1.2114  0.1785
#> 3 0.8074  0.1299
#> 4 1.3184  0.3125
#> 5 0.9899  0.0573
SummariesAll$Gaussian$env[, 8:14]
#>      MAE  MAE_SE      Cor Cor_SE Intercept Intercept_SE
#> 1 401.9890 64.9822  0.8531 0.0851 -196.7199    322.2706
#> 2 561.5175 83.1826  0.4102 0.1064  865.4911    184.9300
#> 3 396.3470 59.0103  0.6941 0.0872  -28.2293    426.7363
#> 4 874.9023 40.0374 -0.0145 0.3099 1388.0837    976.7171
#> 5 564.8448 35.9622  0.4743 0.0442  320.9136    370.9323
#>      Slope
#> 1 1.0014
#> 2 0.4451
#> 3 1.0163
#> 4 0.2495
#> 5 0.8234
SummariesAll$Gaussian$env[, 15:19]
#>      Slope_SE      R2  R2_SE  MAAPE  MAAPE_SE
#> 1  0.2688 0.7568 0.1266 0.3345  0.0704
#> 2  0.0821 0.2135 0.0928 0.3825  0.0416
#> 3  0.3329 0.5122 0.1240 0.2780  0.0428
#> 4  0.6952 0.3843 0.1211 0.4882  0.0531

```



```
#> 5    0.3074 0.2327 0.0428 0.3498    0.0208

# Summaries by Fold
SummariesAll$Gaussian$fold[, 1:8]
#>      Fold      MSE      MSE_SE      RMSE      RMSE_SE      NRMSE      NRMSE_SE
#> 1      1 612080.5 260226.78 712.2547 186.8812 0.8713 0.1819
#> 2      2 535976.9 219912.95 681.7817 154.0028 0.9003 0.1917
#> 3      3 421532.5 150177.05 623.9024 103.7278 2.0261 0.8493
#> 4      4 336513.6 165735.68 533.4182 131.6290 1.1772 0.4547
#> 5      5 561500.5 312850.22 669.0580 194.8178 0.9349 0.1596
#> 6 Global 516453.5 83832.73 710.1459 55.1052 0.9899 0.0573
#>      MAE
#> 1 636.8384
#> 2 584.9749
#> 3 535.4100
#> 4 476.1404
#> 5 560.0810
#> 6 564.8448
SummariesAll$Gaussian$fold[, 9:14]
#>      MAE_SE      Cor Cor_SE Intercept Intercept_SE      Slope
#> 1 164.4052 0.2783 0.4234 1315.2848 1133.2039 0.1350
#> 2 139.5142 0.6156 0.1438 406.4774 284.2862 0.7273
#> 3 76.2590 0.5432 0.1862 946.2454 354.7883 0.3715
#> 4 120.2463 0.3608 0.3202 716.1013 422.6240 0.4514
#> 5 135.7841 0.6308 0.1192 -848.3269 494.4580 1.7052
#> 6 35.9622 0.4743 0.0442 320.9136 370.9323 0.8234
SummariesAll$Gaussian$fold[, 15:19]
#>      Slope_SE      R2      R2_SE      MAAPE      MAAPE_SE
#> 1 0.6951 0.6151 0.2048 0.4402 0.1070
#> 2 0.2153 0.4410 0.1792 0.3706 0.0769
#> 3 0.1605 0.3991 0.1551 0.3734 0.0566
#> 4 0.2631 0.4378 0.1570 0.3358 0.0679
#> 5 0.4340 0.4405 0.1560 0.3340 0.0346
#> 6 0.3074 0.2327 0.0428 0.3498 0.0208
```

Además, los elementos de la lista *HyperparamsAll* contienen las columnas *tree_number*, *node_size*, *shrinkage*, *mse* y *Fold*, donde el valor de la columna *ms* corresponde al costo del modelo para cada combinación de los valores de partición e hiperparámetro, ordenados de menor a mayor dentro de cada partición.

```
# First rows of Hyperparams
head(HyperparamsAll$Gaussian)
#>      trees_number node_size shrinkage      mse Fold
#> 26             300         15      0.10 372385.3   1
#> 19             100          5      0.10 384601.8   1
#> 15             500         10      0.01 387769.0   1
#> 20             300          5      0.10 389870.6   1
#> 23             300         10      0.10 392475.9   1
#> 21             500          5      0.10 395593.9   1
# Last rows of Hyperparams
```

```
tail(HyperparamsAll$Gaussian)
#>      trees_number node_size shrinkage      mse Fold
#> 224           100       10      0.10 452863.8    5
#> 264           300       15      0.10 454758.4    5
#> 242           500       10      0.10 456085.5    5
#> 122           500        5      0.01 460857.3    5
#> 204           300        5      0.10 475979.2    5
#> 194           100        5      0.10 511498.7    5
```

6.5 Ejemplo de con Sparse Kernels con Grid earch y particiones aleatorias

Este ejemplo evalúa un modelo de Máquina Impulsada Generalizada con cinco particiones aleatorias, con 20% los datos para el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de cada partición (los parámetros predeterminados de la función `cv_random`), para una respuesta continua, usando la matriz de diseño de la variable PhenoToy Env, la matriz descrita G anteriormente y la matriz de diseño de la interacción entre estas dos, como predictores; así como el uso de "Grid Search" como un tipo de tuneo para los hiperparámetros *trees_number*, *node_size* y *encogimiento*. Todo esto para los llamados "Sparse Kernel Methods", con las posibles combinaciones entre los tipos de Kernel "Sparse_Gaussian" y "Sparse_Arc_cosine" con las proporciones 0.5,0.6,0.7,0.8,0.9 y 1.

En este ejemplo, el conjunto de datos utilizado es *MaizeToy* y el objetivo es predecir la variable continua *ASI* del *data frame* PhenoToy utilizando la matriz de diseño de la variable PhenoToy Env, la matriz descrita G anteriormente y la matriz de diseño de la interacción entre estos dos, como predictores; por lo que identificamos las variables predictoras y de respuesta como X y y respectivamente.

```
# Load the dataset
load("MaizeToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env, G & GE
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LinexGeno <- Line %*% Geno
LinexGenoxEnv <- model.matrix(~ 0 + LinexGeno:Env)

# Predictor and Response Variables
X <- cbind(Env, LinexGeno, LinexGenoxEnv)
y <- PhenoToy$ASI

dim(X)
#> [1] 90 123
print(y[1:7])
```

```
#> [1] 1.4 1.0 2.0 2.0 1.4 1.3 3.0
typeof(y)
#> [1] "double"
```

Tenga en cuenta que la variable *y* es una variable continua (un vector con elementos de tipo "double"), lo cual es importante para que el modelo se entrene automáticamente para una variable continua.

A diferencia de los ejemplos anteriores, ahora buscamos evaluar el modelo para cada una de las posibles combinaciones entre los tipos de Kernel "Sparse_Gaussian" y "Sparse_Arc_cosine" con las proporciones 0.5, 0.6, 0.7, 0.8, 0.9 y 1. Por eso creamos un vector llamado *kernels* en el que indicamos los tipos de kernels que queremos aplicar a los de la matriz *X* y otro vector llamado *lines_proportions*. Además, creamos las listas vacías *PredictionsAll*, *TimesAll*, *HyperparamsAll* y *SummariesAll* que se utilizarán para guardar las predicciones, los tiempos de ejecución, los hiperparámetros y los resúmenes de cada modelo entrenado, es decir, para cada combinación entre tipo de kernel y proporción de líneas utilizadas, que a su vez servirán para guardar los valores observados y predichos en cada ambiente y para posteriormente evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
kernels <- c("Sparse_Gaussian", "Sparse_Arc_cosine")
lines_proportions <- c(0.5, 0.6, 0.7, 0.8, 0.9, 1)

# Empty lists that will contain Predictions,
# Times of execution & Summaries for each type of kernel
PredictionsAll <- list()
TimesAll <- list()
HyperparamsAll <- list()
SummariesAll <- list()
```

Posteriormente, se seguirá el siguiente proceso **para cada tipo de Kernel y para cada proporción de líneas**:

1. El conjunto de tipos de kernel se aplica a la matriz de datos *X*, asignando el valor numérico a la 2ª fila. El argumento *arc_cosine_deep* y la proporción de líneas establecen el valor en el argumento de proporción de filas.
2. Luego realizamos cinco particiones aleatorias, con 80% de los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función *cv_random*.
3. *Predictions*, *Times* e *Hyperparams* que se utilizarán para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función *gs_summaries*, además de guardar los tiempos de ejecución de cada modelo entrenado para cada partición.
4. **Para cada partición:**
 1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;

2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo los valores 100, 300 y 500 para el hiperparámetro *tree_number*, los valores 5, 10 y 15 para el hiperparámetro *node_size* y los valores 0.001, 0.01 y 0.1 para el hiperparámetro de contracción, con "Grid Search" como tipo de ajuste . (parámetro predeterminado de *tune_type*);
 3. Con el modelo obtenido en (2), se predice la variable respuesta y en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
 4. Identificación de las predicciones del conjunto de prueba: Se crea el data frame *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba. Además, cada fila de *FoldPrediction* se agrega al data frame *Predictions**.
 5. Identificación del tiempo de ejecución del entrenamiento del modelo obtenido en (2): Se crea el data frame *FoldTime* que contiene la columna que especifica el tipo de *Kernel* utilizado para entrenar el modelo, el número de *Fold* y la columna *Minutos* que indica el tiempo de ejecución, en minutos, del entrenamiento del modelo obtenido en (2). Además, cada fila de *FoldTime* se agrega al data frame *Times*.
 6. Identificación de hiperparámetros; Se crea el data frame *HyperparamsFold* que contiene las columnas *tree_number*, *node_size*, *shrinkage*, *mse* y *Fold*, donde *mse* corresponde al costo del modelo para cada combinación de los hiperparámetros especificados y el número de *Fold*. Además, cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
- *Predictions* contiene las columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo los hiperparámetros óptimos (entre las posibles combinaciones de estos) que minimizan la función de costo con el tipo de tuneo "Optimización bayesiana", correspondiente al formato necesario para utilizar la función *gs_summaries* en estas predicciones en el caso de variables continuas.
5. *Predictions* y con la ayuda de la función *gs_summaries*, evaluamos la capacidad predictiva del modelo entrenado para el tipo de kernel especificado. La función *gs_summaries* devuelve una lista que identificamos como *summaries* y que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.
 6. Finalmente, se crea un elemento con el nombre de kernel especificado en cada una de las listas *PredictionsAll*, *TimesAll*, *HyperparamsAll* y *SummariesAll*, que corresponden a los marcos de datos de las listas *Predictions*, *Times*, *Hyperparams* y *summaries*, respectivamente.

```
for (kernel in kernels) {
  cat("*** Kernel:", kernel, "***\n")
  for (line_proportion in lines_proportions) {
    cat("\t*** Line_Proportion:", line_proportion, "***\n")
  }
}
```

```

# Compute the kernel
X <- kernelize(
  X,
  kernel = kernel,
  arc_cosine_deep = 2,
  rows_proportion = line_proportion
)

# Random Partition
set.seed(2022)
folds <- cv_random(
  records_number = nrow(X),
  folds_number = 5,
  testing_proportion = 0.2
)

# Empty data frames that will contain Predictions, Times
# of execution & Summaries for each partition
Predictions <- data.frame()
Times <- data.frame()
Hyperparams <- data.frame()

for (i in seq_along(folds)) {
  cat("\t*** Fold:", i, "***\n")
  fold <- folds[[i]]

  # Identify the training and testing sets
  X_training <- X[fold$training, ]
  X_testing <- X[fold$testing, ]
  y_training <- y[fold$training]
  y_testing <- y[fold$testing]

  # Model training
  model <- generalized_boosted_machine(
    x = X_training,
    y = y_training,

    # Specify the hyperparameters values
    trees_number = c(100, 300, 500),
    node_size = c(5, 10, 15),
    max_depth = 5,
    shrinkage = c(0.001, 0.01, 0.1),
    sampled_records_proportion = 0.6,
    tune_type = "grid_search",
    tune_grid_proportion = 0.8,
    # In this example the iterations wont be shown
    verbose = FALSE
  )
}

```

```

# Testing Predictions
predictions <- predict(model, X_testing)

# Predictions for the Fold Fold
Predictions <- data.frame(
  Fold = i,
  Line = PhenoToy$Line[fold$testing],
  Env = PhenoToy$Env[fold$testing],
  Observed = y_testing,
  Predicted = predictions$predicted
)
Predictions <- rbind(Predictions, FoldPredictions)

# Execution times
FoldTime <- data.frame(
  kernel = kernel,
  Fold = i,
  Minutes = as.numeric(model$execution_time, units = "mins")
)
Times <- rbind(Times, FoldTime)

# Hyperparams for the Fold
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)
}
# Summaries of the Folds
summaries <- gs_summaries(Predictions)
str_line <- paste("Line_Proportion:", line_proportion)

# Predictions, Times of execution & Summaries for the
# specified Kernel
PredictionsAll[[kernel]][[str_line]] <- Predictions
TimesAll[[kernel]][[str_line]] <- Times
HyperparamsAll[[kernel]][[str_line]] <- Hyperparams
SummariesAll[[kernel]][[str_line]] <- summaries
}
}
#> *** Kernel: Sparse_Gaussian ***
#> *** Line_Proportion: 0.5 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.6 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***

```

```

#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.7 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.8 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.9 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 1 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Kernel: Sparse_Arc_cosine ***
#> *** Line_Proportion: 0.5 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.6 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.7 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.8 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***

```

```
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 0.9 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
#> *** Line_Proportion: 1 ***
#> *** Fold: 1 ***
#> *** Fold: 2 ***
#> *** Fold: 3 ***
#> *** Fold: 4 ***
#> *** Fold: 5 ***
```

Recordando que este proceso se realizó para cada combinación de tipo de kernel y relación de línea especificada, cada una de las listas *PredictionsAll*, *TimesAll*, *HyperparamsAll* y *SummariesAll* contiene las predicciones, los tiempos de ejecución, las combinaciones de hiperparámetros (en este caso *alpha*) y los resúmenes, respectivamente, para cada combinación entre el tipo de kernel y la proporción de *líneas* aplicadas a la matriz de datos *X*. A modo de ejemplo, a continuación se muestran los resultados obtenidos para el tipo de kernel "Sparse_Arc_cosine" y "Line_Proportion 1":

```
# Predictions for the Sparse_Arc_cosine Kernel
head(PredictionsAll$Sparse_Arc_cosine$`Line_Proportion: 1`)
#>   Fold      Line Env Observed Predicted
#> 1     5 CKDHL0129 KAK      -0.3  1.558582
#> 2     5 CKDHL0647 KTI       2.3  1.985260
#> 3     5 CKDHL0054 KAK       1.8  1.533947
#> 4     5 CKDHL0647 KAK       0.2  1.589915
#> 5     5 CKDHL0052 EBU       1.7  2.011850
#> 6     5 CKDHL0437 KTI       1.7  1.457221
# Times of execution for the Sparse_Arc_cosine Kernel
TimesAll$Sparse_Arc_cosine$`Line_Proportion: 1`
#>           kernel Fold  Minutes
#> 1 Sparse_Arc_cosine    1 0.1110234
#> 2 Sparse_Arc_cosine    2 0.1199914
#> 3 Sparse_Arc_cosine    3 0.1154964
#> 4 Sparse_Arc_cosine    4 0.1199468
#> 5 Sparse_Arc_cosine    5 0.1197647
# Elements of SummariesAll
names(SummariesAll)
#> [1] "Sparse_Gaussian" "Sparse_Arc_cosine"
# Elements of summaries for Sparse_Arc_cosine Kernel
names(SummariesAll$Sparse_Arc_cosine)
#> [1] "Line_Proportion: 0.5" "Line_Proportion: 0.6"
#> [3] "Line_Proportion: 0.7" "Line_Proportion: 0.8"
#> [5] "Line_Proportion: 0.9" "Line_Proportion: 1"
names(SummariesAll$Sparse_Arc_cosine$`Line_Proportion: 1`)
#> [1] "line" "env" "fold"
```



```

# Summaries by Line
head(SummariesAll$Sparse_Arc_cosine$`Line_Proprtion: 1`$line)
#>      Line Observed Predicted Difference
#> 1 CKDHL0206      2.1      2.0938      0.0062
#> 2 CKDHL0491      2.0      2.0677      0.0677
#> 3 CKDHL0529      1.3      1.4462      0.1462
#> 4 CKDHL0437      1.7      1.4572      0.2428
#> 5 CKDHL0050      1.3      1.5543      0.2543
#> 6 CKDHL0054      1.8      1.5339      0.2661

# Summaries by Enviroment
SummariesAll$Sparse_Arc_cosine$`Line_Proprtion: 1`$env[, 1:7]
#>      Env      MSE MSE_SE      RMSE RMSE_SE  NRMSE
#> 1      EBU      0.7607      NA      0.8722      NA 0.9945
#> 2      KAK      2.2736      NA      1.5078      NA 0.8990
#> 3      KTI      0.9267      NA      0.9627      NA 1.2977
#> 4 ALIYARNAGAR_R15 627573.7512      NA 792.1955      NA 0.9591
#> 5 ICRISAT_PR15-16 188102.4708      NA 433.7078      NA 1.3867
#> 6      ICRISAT_R15 328532.7892      NA 573.1778      NA 0.9737
#> 7      JALGOAN_R15 1507005.8300      NA 1227.6017      NA 0.9614
#> 8      Global 479921.7249      NA 692.7638      NA 0.5883
#>      NRMSE_SE
#> 1      NA
#> 2      NA
#> 3      NA
#> 4      NA
#> 5      NA
#> 6      NA
#> 7      NA
#> 8      NA
SummariesAll$Sparse_Arc_cosine$`Line_Proprtion: 1`$env[, 8:14]
#>      MAE MAE_SE      Cor Cor_SE Intercept Intercept_SE
#> 1  0.6189      NA -0.1495      NA      2.6899      NA
#> 2  1.2071      NA -0.0960      NA     12.0438      NA
#> 3  0.7213      NA -0.0288      NA      2.3943      NA
#> 4 652.0991      NA -0.5960      NA 20374.9784      NA
#> 5 396.2854      NA  0.3702      NA   761.4247      NA
#> 6 495.9420      NA -0.2435      NA 3355.9905      NA
#> 7 973.4627      NA  0.5616      NA   448.9254      NA
#> 8 377.2817      NA  0.8411      NA     3.5397      NA
#>      Slope
#> 1 -0.4031
#> 2 -6.7591
#> 3 -0.0921
#> 4 -13.2924
#> 5  0.3051
#> 6 -1.3830
#> 7  1.1837
#> 8  1.2689
SummariesAll$Sparse_Arc_cosine$`Line_Proprtion: 1`$env[, 15:19]

```

```

#>   Slope_SE      R2 R2_SE  MAAPE MAAPE_SE
#> 1      NA 0.0223    NA 0.2879      NA
#> 2      NA 0.0092    NA 0.7184      NA
#> 3      NA 0.0008    NA 0.2592      NA
#> 4      NA 0.3552    NA 0.4269      NA
#> 5      NA 0.1370    NA 0.3271      NA
#> 6      NA 0.0593    NA 0.3454      NA
#> 7      NA 0.3154    NA 0.3966      NA
#> 8      NA 0.7075    NA 0.3553      NA

# Summaries by Fold
SummariesAll$Sparse_Arc_cosine$`Line_Proprtion: 1`$fold[, 1:7]
#>      Fold      MSE MSE_SE      RMSE RMSE_SE  NRMSE NRMSE_SE
#> 1      5 378745.5 207255 432.8608 178.595 1.0674 0.0724
#> 2 Global 479921.7      NA 692.7638      NA 0.5883      NA
SummariesAll$Sparse_Arc_cosine$`Line_Proprtion: 1`$fold[, 8:14]
#>      MAE MAE_SE      Cor Cor_SE Intercept Intercept_SE
#> 1 360.0481 143.7941 -0.0260 0.1459 3565.4924      2837.65
#> 2 377.2817      NA 0.8411      NA 3.5397      NA
#>      Slope
#> 1 -2.9201
#> 2 1.2689
SummariesAll$Sparse_Arc_cosine$`Line_Proprtion: 1`$fold[, 15:19]
#>   Slope_SE      R2 R2_SE  MAAPE MAAPE_SE
#> 1 1.9896 0.1285 0.0563 0.3945 0.0583
#> 2      NA 0.7075      NA 0.3553      NA

```

Además, los elementos de la lista *HyperparamsAll* contienen las columnas *tree_number*, *node_size*, *shrinkage*, *mse* y *Fold*, donde el valor de la columna *mse* corresponde al costo del modelo para cada combinación de los valores de partición e hiperparámetro, ordenados de menor a mayor dentro de cada partición.

```

# First rows of Hyperparams
head(HyperparamsAll$Sparse_Arc_cosine$`Line_Proprtion: 1`)
#>   trees_number node_size shrinkage      mse Fold
#> 7           100        15    0.001 1.104567 1
#> 4           100        10    0.001 1.106264 1
#> 1           100         5    0.001 1.107948 1
#> 5           300        10    0.001 1.111888 1
#> 6           500        10    0.001 1.112149 1
#> 16          100        15    0.010 1.123677 1

# Last rows of Hyperparams
tail(HyperparamsAll$Sparse_Arc_cosine$`Line_Proprtion: 1`)
#>   trees_number node_size shrinkage      mse Fold
#> 264          300        15    0.1 1.161714 5
#> 193          100         5    0.1 1.286859 5
#> 233          300        10    0.1 1.326050 5
#> 273          500        15    0.1 1.364316 5
#> 243          500        10    0.1 1.543954 5
#> 215          500         5    0.1 1.632892 5

```

In addition, the *HyperparamsAll* list items contain the columns *trees_number*, *node_size*, *shrinkage*, *mse* and *Fold*, where the value of the *mse* column corresponds to the cost of the model for each combination of the partition and hyperparameter values, ordered from smallest to largest. largest within each partition.

```
# First rows of Hyperparams
head(HyperparamsAll$Sparse_Arc_cosine$`Line_Proprtion: 1`)
#>   trees_number node_size shrinkage      mse Fold
#> 7           100        15    0.001 1.104567    1
#> 4           100        10    0.001 1.106264    1
#> 1           100         5    0.001 1.107948    1
#> 5           300        10    0.001 1.111888    1
#> 6           500        10    0.001 1.112149    1
#> 16          100        15    0.010 1.123677    1
# Last rows of Hyperparams
tail(HyperparamsAll$Sparse_Arc_cosine$`Line_Proprtion: 1`)
#>   trees_number node_size shrinkage      mse Fold
#> 264           300        15     0.1 1.161714    5
#> 193           100         5     0.1 1.286859    5
#> 233           300        10     0.1 1.326050    5
#> 273           500        15     0.1 1.364316    5
#> 243           500        10     0.1 1.543954    5
#> 215           500         5     0.1 1.632892    5
```

7 Support vector machine methods

7.1 Ejemplo con respuesta continua con Grid search y particiones aleatorias solo *G* en el predictor

Este ejemplo evalúa un modelo de máquina de vectores de soporte con cinco particiones aleatorias, cn 20% los datos para el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de cada partición (los parámetros predeeterminados de la función *cv_random*), para una respuesta continua, usand solo la matriz *G* (Line Design Matriz que contiene información genómica) como predictor y utiliza "Grid Search" como ipo de tuneo para los hiperparámetros *grado*, *gamma* y *coef0*.

En este ejemplo, el conjunto de datos utilizado es *ChickpeaToy* y el objetivo es predecir la variable continua *AvePlantHeight* del marco de datos *PhenoToy* utilizando la matriz *G* descrita anteriormente como predictor; por lo que identificamos las variables predictoras y de respuesta como *X* y *y* respectivamente.

```
# Load the data
load("ChickpeaToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of G
```

```

Line <- model.matrix(~ 0 + Line, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %**% Geno

# Predictor and Response Variables
X <- LineG
y <- PhenoToy$AvePlantHeight

# Note that y is a continuous numeric vector
class(y)
#> [1] "numeric"
typeof(y)
#> [1] "double"

```

Tenga en cuenta que la variable de respuesta *y* es una variable continua (un vector con elemento de tipo "doble"), lo cual es importante para que el modelo se entrene automáticamente para este tipo de variable.

Posteriormente, realizamos cinco particiones aleatorias, con 80% los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función `cv_random` (con los parámetros por defecto). Además, creamos los dataframes vacíos `Predictions` e `Hyperparams` que se utilizarán para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función `gs_summaries`.

```

# Set seed for reproducible results
set.seed(2022)
folds <- cv_random(records_number = nrow(X))

# A data frame that will contain the variables:
Predictions <- data.frame()
Hyperparams <- data.frame()

```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo "polinomial" como *tipo de núcleo*, los valores 1, 5 y 10 para el hiperparámetro *gamma*, los valores 1, 2 y 3 para el hiperparámetro *grado*, y los valores 0 y 5 para el hiperparámetro *coef0*, con "Grid Search" como *tipo de tuneo* (parámetro *predeterminado de tune_type*). Tenga en cuenta que, con la combinación de hiperparámetro *gamma* = 1, *degre* = 1* y *coef0* = 0, el núcleo lineal se considera como un posible núcleo de transformación. Cabe señalar que estos no son los únicos hiperparámetros ajustables en el modelo;

3. Con el modelo obtenido en (2), se predice la variable respuesta *AvePlantHeight* en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba.
 - b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*.
5. Identificación de hiperparámetros:
 - a. *HyperparamsFold* que contiene las columnas *grado*, *gamma*, *coef0*, *mse* y *Fold*, donde *mse* es la precisión del modelo para cada combinación de los hiperparámetros especificados y el número de *Fold*.
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
6. Se muestran los hiperparámetros óptimos del modelo obtenido en (2).

```
# Model training and predictions of the ith partition
```

```
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")
  fold <- folds[[i]]
```

```
  # Identify the training and testing sets
```

```
  X_training <- X[fold$training, ]
  X_testing <- X[fold$testing, ]
  y_training <- y[fold$training]
  y_testing <- y[fold$testing]
```

```
  # Model training
```

```
  model <- support_vector_machine(
    x = X_training,
    y = y_training,
    kernel = "polynomial",
```

```
    # Specify the hyperparameters
```

```
    gamma = c(1, 5, 10),
    degree = c(1, 2, 3),
    coef0 = c(0, 5),
    tune_type = "grid_search",
    # In this example the iterations wont be shown
    verbose = FALSE
```

```
)
```

```
  # Prediction of the test set
```

```
  predictions <- predict(model, X_testing)
```

```

# Prediction of the test set
predictions <- predict(model, X_testing)

# Predictions for the Fold Fold
FoldPredictions <- data.frame(
  Fold = i,
  Line = PhenoToy$Line[fold$testing],
  Env = PhenoToy$Env[fold$testing],
  Observed = y_testing,
  Predicted = predictions$predicted
)
Predictions <- rbind(Predictions, FoldPredictions)

# Hyperparams
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)

# Best hyperparams of the model
cat("*** Optimal hyperparameters: ***\n")
print(model$best_hyperparams)
}
#> *** Fold: 1 ***
#> *** Optimal hyperparameters: ***
#> $degree
#> [1] 2
#>
#> $gamma
#> [1] 1
#>
#> $coef0
#> [1] 5
#>
#> $mse
#> [1] 123.1633
#>
#> *** Fold: 2 ***
#> *** Optimal hyperparameters: ***
#> $degree
#> [1] 3
#>
#> $gamma
#> [1] 10
#>
#> $coef0
#> [1] 5
#>
#> $mse
#> [1] 109.5256
#>

```

```

#> *** Fold: 3 ***
#> *** Optimal hyperparameters: ***
#> $degree
#> [1] 1
#>
#> $gamma
#> [1] 1
#>
#> $coef0
#> [1] 5
#>
#> $mse
#> [1] 116.5544
#>
#> *** Fold: 4 ***
#> *** Optimal hyperparameters: ***
#> $degree
#> [1] 1
#>
#> $gamma
#> [1] 1
#>
#> $coef0
#> [1] 5
#>
#> $mse
#> [1] 144.7736
#>
#> *** Fold: 5 ***
#> *** Optimal hyperparameters: ***
#> $degree
#> [1] 2
#>
#> $gamma
#> [1] 5
#>
#> $coef0
#> [1] 0
#>
#> $mse
#> [1] 135.5366

```

El data frame Predictions contiene las columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba para cada partición, donde las predicciones se realizan eligiendo los hiperparámetros óptimos (entre los posibles valores especificados de estos) que minimizan la función de costo con el tipo de tuneo "Grid Search", correspondiente al formato necesario para utilizar la función *gs_summaries* sobre estas predicciones en el caso de variables continuas.

La función `gs_summaries` devuelve unalista que contiene tres marcos de daos correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```
head(Predictions)
#>   Fold      Line Env Observed Predicted
#> 1     1 ICCV97301   6 65.33333  46.22952
#> 2     1 ICCV04103   1 47.56667  50.44068
#> 3     1 ICCV05109   4 52.66667  44.56186
#> 4     1 ICCV00402   7 38.00000  53.60103
#> 5     1 ICCV09114   4 51.33333  40.56444
#> 6     1 ICCV03102   2 52.30000  43.90145
unique(Predictions$Fold)
#> [1] 1 2 3 4 5
# Summaries
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env"  "fold"
# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted Difference
#> 1 ICCV04312  48.4417  48.8928    0.4511
#> 2 ICCV09118  43.3083  43.8347    0.5264
#> 3 ICCV03109  44.3250  43.6299    0.6951
#> 4 ICCV03309  48.6000  49.3179    0.7179
#> 5 ICCV10316  47.4800  48.3318    0.8518
#> 6 ICCV05307  48.4250  47.4226    1.0024

# Summaries by Environment
summaries$env[, 1:7]
#>      Env      MSE MSE_SE      RMSE RMSE_SE  NRMSE NRMSE_SE
#> 1     1 133.9711 12.1319 11.5315  0.4990 2.2142  0.2286
#> 2     2 228.8790 24.0220 15.0423  0.8073 2.7426  0.4820
#> 3     4  99.0279 11.3722  9.8850  0.5732 3.1468  0.7677
#> 4     5  75.7551 18.3454  8.2487  1.3887 2.3232  0.3340
#> 5     6 179.1657 20.8968 13.2788  0.8423 2.3767  0.3472
#> 6     7  57.8016 11.8498  7.4578  0.7387 2.2184  0.1309
#> 7 Global 117.4334  8.1866 10.8100  0.3796 1.3595  0.0516
summaries$env[, 8:14]
#>      MAE MAE_SE      Cor Cor_SE Intercept Intercept_SE  Slope
#> 1 10.2879 0.6260  0.1049 0.2253  26.8901    23.1197  0.2571
#> 2 14.2996 0.9364  0.5774 0.2038  -4.9334    38.3653  1.3734
#> 3  9.0451 0.5137  0.3784 0.1460  34.9485     8.0318  0.4289
#> 4  7.4727 1.3193  0.5173 0.1511   9.5222    19.0309  0.8063
#> 5 11.4175 0.9137 -0.4103 0.1338  80.9165    12.8941 -0.5150
#> 6  6.8727 0.7932  0.7160 0.0918  -2.7647    19.0949  0.9011
#> 7  9.3561 0.3857 -0.3282 0.0373  78.0954     3.0798 -0.6107
summaries$env[, 15:19]
#>      Slope_SE      R2  R2_SE  MAAPE MAAPE_SE
```



```

#> 1  0.4661 0.2140 0.0502 0.2607  0.0167
#> 2  0.7953 0.4994 0.1360 0.2288  0.0137
#> 3  0.1785 0.2285 0.0928 0.1627  0.0090
#> 4  0.3746 0.3532 0.1815 0.1802  0.0326
#> 5  0.2371 0.2400 0.1017 0.1902  0.0159
#> 6  0.3700 0.5463 0.1263 0.1629  0.0203
#> 7  0.0547 0.1133 0.0250 0.1901  0.0071

# Summaries by Fold
summaries$fold[, 1:8]
#>      Fold      MSE MSE_SE    RMSE RMSE_SE  NRMSE NRMSE_SE
#> 1      1 131.0179 29.2932 11.1210  1.2117 2.9079  0.6392
#> 2      2 122.5273 36.8365 10.1713  1.9530 2.5791  0.1935
#> 3      3 143.4187 40.4167 11.3362  1.7268 2.3692  0.2271
#> 4      4 130.0817 26.7126 11.0675  1.2322 2.6761  0.4294
#> 5      5 118.4546  9.8020 10.8408  0.4319 2.0287  0.2837
#> 6 Global 117.4334  8.1866 10.8100  0.3796 1.3595  0.0516
#>      MAE
#> 1  9.9987
#> 2  9.3767
#> 3 10.4072
#> 4  9.9723
#> 5  9.7413
#> 6  9.3561

```

Además, Hyperparams contiene columnas de *grado*, *gamma*, *coef0*, *mse* *Fold*, donde el valor en la columna *mse* corresponde al costo del modelo par cada combinación de hiperparámetros y partición, ordenado de menor a mayor dentro e cada partición.

```

# First rows of Hyperparams
head(Hyperparams)
#>   degree gamma coef0      mse Fold
#> 11      2      1      5 123.1633   1
#> 14      2      5      5 123.1732   1
#> 17      2     10      5 123.1865   1
#> 2       2      1      0 123.2125   1
#> 8       2     10      0 123.2125   1
#> 5       2      5      0 123.2125   1

# Last rows of Hyperparams
tail(Hyperparams)
#>   degree gamma coef0      mse Fold
#> 74      1     10      0 136.0731   5
#> 44      1      5      0 136.0731   5
#> 134     1      5      5 136.0731   5
#> 164     1     10      5 136.0731   5
#> 116     1      1      0 136.0731   5
#> 104     1      1      5 136.0731   5

```

7.2 Ejemplo con respuesta binaria con Grid search y validación cruzada 7-fold con *Env* + *G* el predictor

Este ejemplo evalúa un modelo de máquina de vectores de soporte con validación cruzada 7-fold, para una respuesta binaria, utilizando el efecto de ambiente y la matriz *G* como predictores, además de "búsqueda de cuadrícula" como tipo de tuneo para los hiperparámetros *grado*, *gamma* y *coef0*.

En este ejemplo, el conjunto de datos utilizado es *EYTTToy* y el objetivo es predecir la variable binaria y_{bin} , que es una transformación de la variable *PhenoTy STMT*, indicando con la función *ntile* si la respuesta es mayor que la mediana de esta variable o no, utilizando el matriz de diseño de la variable *PhenoToy Env* y la matriz, *G* descrita anteriormente, como predictores; por lo que identificamos las variables predictoras y de respuesta como *X* y y_{bin} respectivamente.

```
# Load the data
load("EYTTToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env &
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %**% Geno

# Predictor and Response Variables
# Predictor and Response Variables
X <- cbind(Env, LineG)
y_bin <- BurStMisc::ntile(PhenoToy$DTMT, 2, result = "factor")
#> Warning in BurStMisc::ntile(PhenoToy$DTMT, 2, result =
#> "factor"): common values across groups: 1, 2
```

Tenga en cuenta que la variable de respuesta y_{bin} es un factor con solo dos niveles (o categorías), lo cual es importante para que el modelo se entrene automáticamente para una variable binaria. Por esta razón, es importante factorizar esas variables de respuesta binarias o categóricas antes de usar la función **support_vector*.

Posteriormente hacemos las particiones correspondientes a 7-Fold CV, con la ayuda de la función *cv_kfold*. Además, creamos los data frames vacíos *Predictions* e *Hyperparams* que se utilizarán para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
# Set seed for reproducible results
set.seed(2022)
folds <- cv_kfold(records_number = nrow(X), k = 7)
```

```
# A data frame that will contain the variables:
```

```
Predictions <- data.frame()
```

```
Hyperparams <- data.frame()
```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo “polinomial” como *tipo de kernel*, los valores 1, 5 y 10 para el hiperparámetro *gamma* y los valores 50 100, 150 y 200 para el hiperparámetro *coef0*, con “Grid Search” como tipo de ajuste (parámetro por *tune_type* por defecto). Cabe sealar que estos no son los únicos hiperparámetros ajustables en el modelo;
3. Con el modelo obtenido en (2) y_{bin} se predice la variable respuesta en el conjunto de prueba, con el fin de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: *Fold* number, *Line*, *Env*, *Observed*, *Predicted*, 1 y 2 para cada elemento del conjunto de prueba. Tenga en cuenta que, a diferencia del ejemplo anterior, ahora tenemos dos columnas adicionales correspondientes a las probabilidades asociadas con cada elemento correspondiente a esa categoría.
 - b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*. Con esto, *Predictions* se formatea para que sea un argumento de la función *gs_summaries*.
5. Identificación de hiperparámetros:
 - a. *HyperparamsFold* que contiene las columnas *tree_number*, *node_size*, *accuracy* y *Fold*, donde *accuracy* es la precisión del modelo para cada combinación de los hiperparámetros especificados y el número de *Fold*.
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
6. Se muestran los hiperparámetros óptimos.

```
# Model training and predictions of the ith partition
```

```
for (i in seq_along(folds)) {  
  cat("*** Fold:", i, "***\n")  
  fold <- folds[[i]]
```

```
# Identify the training and testing sets
```

```
X_training <- X[fold$training, ]
```

```
X_testing <- X[fold$testing, ]
```

```

y_training <- y_bin[fold$training]
y_testing <- y_bin[fold$testing]

# Model training
model <- support_vector_machine(
  x = X_training,
  y = y_training,
  kernel = "polynomial",

  # Specify the hyperparameters
  gamma = c(1, 5, 10),
  coef0 = c(50, 100, 150, 200),
  tune_type = "grid_search",
  # In this example the iterations wont be shown
  verbose = FALSE
)

# Prediction of the test set
predictions <- predict(model, X_testing)

# Predictions for the Fold Fold
FoldPredictions <- cbind(
  data.frame(
    Fold = i,
    Line = PhenoToy$Line[fold$testing],
    Env = PhenoToy$Env[fold$testing],
    Observed = y_testing,
    Predicted = predictions$predicted
  ),
  predictions$probabilities
)

Predictions <- rbind(Predictions, FoldPredictions)

# Hyperparams
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)

# Best hyperparams of the model
cat("*** Optimal hyperparameters: ***\n")
print(model$best_hyperparams)
}
#> *** Fold: 1 ***
#> *** Optimal hyperparameters: ***
#> $gamma
#> [1] 1
#>
#> $coef0

```

```

#> [1] 150
#>
#> $accuracy
#> [1] 0.8728571
#>
#> *** Fold: 2 ***
#> *** Optimal hyperparameters: ***
#> $gamma
#> [1] 1
#>
#> $coef0
#> [1] 150
#>
#> $accuracy
#> [1] 0.912381
#>
#> *** Fold: 3 ***
#> *** Optimal hyperparameters: ***
#> $gamma
#> [1] 5
#>
#> $coef0
#> [1] 150
#>
#> $accuracy
#> [1] 0.9114286
#>
#> *** Fold: 4 ***
#> *** Optimal hyperparameters: ***
#> $gamma
#> [1] 1
#>
#> $coef0
#> [1] 150
#>
#> $accuracy
#> [1] 0.9033333
#>
#> *** Fold: 5 ***
#> *** Optimal hyperparameters: ***
#> $gamma
#> [1] 1
#>
#> $coef0
#> [1] 50
#>
#> $accuracy
#> [1] 0.8947619
#>
#> *** Fold: 6 ***

```

```

#> *** Optimal hyperparameters: ***
#> $gamma
#> [1] 1
#>
#> $coef0
#> [1] 100
#>
#> $accuracy
#> [1] 0.8742857
#>
#> *** Fold: 7 ***
#> *** Optimal hyperparameters: ***
#> $gamma
#> [1] 1
#>
#> $coef0
#> [1] 100
#>
#> $accuracy
#> [1] 0.8914286

```

Predictions contiene las columnas *Fold*, *Line*, *Env*, *Observed*, *Predicted*, 1 y 2 para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo los hiperparámetros óptimos (entre las posibles combinaciones de sus valores propuestos) que minimizan la función de coste con el tipo de sintonía "Grid Search", correspondiente al formato necesario para utilizar la función *gs_summaries* sobre estas predicciones en el caso de variables binarias.

La función *gs_summaries* devuelve una lista que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fod*.

```

head(Predictions)
#>      Fold      Line      Env Observed Predicted      1
#> 7      1 GID7625106 Flat5IR      1      1 0.839567580
#> 12     1 GID7625276 FlatDrip      1      1 0.992210659
#> 13     1 GID7625985 Bed5IR      1      2 0.206579259
#> 18     1 GID7626366 EHT      2      2 0.005132066
#> 24     1 GID7626381 FlatDrip      1      1 0.874248611
#> 32     1 GID7626446 FlatDrip      1      1 0.845232922
#>
#>      2
#> 7 0.160432420
#> 12 0.007789341
#> 13 0.793420741
#> 18 0.994867934
#> 24 0.125751389
#> 32 0.154767078
unique(Predictions$Fold)
#> [1] 1 2 3 4 5 6 7
# Summaries
summaries <- gs_summaries(Predictions)

```

```

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted      X1      X2
#> 1  GID7462121      1      1 0.7594 0.2406
#> 2  GID7625106      1      1 0.7385 0.2615
#> 3  GID7625276      1      1 0.5862 0.4138
#> 4  GID7625985      1      2 0.3243 0.6757
#> 5  GID7626366      2      2 0.2576 0.7424
#> 6  GID7626381      2      2 0.3022 0.6978

# Summaries by Environment
summaries$env
#>      Env      PCCC PCCC_SE      Kappa Kappa_SE BrierScore
#> 1  Bed5IR 0.9643 0.0357 0.9286 0.0714 0.1457
#> 2      EHT 1.0000 0.0000      NaN      NA 0.0335
#> 3  Flat5IR 0.9439 0.0381 0.8391 0.0873 0.1394
#> 4  FlatDrip 0.9167 0.0546 0.0000 0.0000 0.1927
#> 5   Global 0.9591 0.0145 0.9134 0.0307 0.1494
#>      BrierScore_SE
#> 1      0.0426
#> 2      0.0177
#> 3      0.0432
#> 4      0.0767
#> 5      0.0293

# Summaries by Fold
summaries$fold
#>      FoLd      PCCC PCCC_SE      Kappa Kappa_SE BrierScore
#> 1      1 0.9375 0.0625 0.5000      NA 0.1036
#> 2      2 1.0000 0.0000 1.0000 0.0000 0.0667
#> 3      3 0.8542 0.0859 0.5000 0.2500 0.2612
#> 4      4 1.0000 0.0000 1.0000      NA 0.0860
#> 5      5 0.9375 0.0625 0.6667 0.2887 0.1400
#> 6      6 1.0000 0.0000 1.0000 0.0000 0.0910
#> 7      7 0.9643 0.0357 0.8478 0.1076 0.1461
#> 8 Global 0.9591 0.0145 0.9134 0.0307 0.1494
#>      BrierScore_SE
#> 1      0.0800
#> 2      0.0326
#> 3      0.0681
#> 4      0.0260
#> 5      0.1210
#> 6      0.0384
#> 7      0.0710
#> 8      0.0293

```

Además, Hyperparams contiene columnas *gamma*, *coef0*, *accuracy* y *Fold*, donde el valor de la columna de *precisión* corresponde al costo del modelo para cada combinación de valores de partición e hiperparámetro, ordenados de menor a mayor dentro de cada partición.

```
# First rows of Hyperparams
head(Hyperparams)
#>   gamma coef0 accuracy Fold
#> 7      1    150 0.8728571    1
#> 8      5    150 0.8728571    1
#> 1      1     50 0.8533333    1
#> 11     5    200 0.8533333    1
#> 10     1    200 0.8528571    1
#> 4      1    100 0.8528571    1
# Last rows of Hyperparams
tail(Hyperparams)
#>   gamma coef0 accuracy Fold
#> 126    10    200 0.8038095    7
#> 56     5    100 0.7938095    7
#> 96     10    150 0.7757143    7
#> 26     5     50 0.7561905    7
#> 66     10    100 0.7466667    7
#> 36     10     50 0.6890476    7
```

7.3 Ejemplo de con respuesta categórica con optimización bayesiana con línea de partición aleatoria *Env* + *G* + *GE* dentro del predictor

Este ejemplo evalúa un modelo de máquina de vectores de soporte con cinco particiones aleatorias del conjunto de líneas, con 80% las líneas para el conjunto de entrenamiento y 20% para el conjunto de entrenamiento dentro de cada partición (los parámetros predeterminados de la función *cv_random*), para una respuesta categórica, usando el efecto de el ambiente, la matriz *G* y la interacción entre estos dos como predictores, además de utilizar la "Optimización Bayesiana" como un tipo de tuneo de hiperparámetros.

En este ejemplo, el conjunto de dato utilizado es *GroundnutToy* y busca predecir la variable categórica *y*, que es una transformación de la variable *YPH* del data frame *PhenoToy* utilizando la función *ntile*, utilizando la matriz de diseño de la variable *PhenoToy Env*, la matriz *G* descrito anteriormente y la matriz de diseño de la interacción entre estos dos, como predictores; por lo que identificamos las variables predictoras y de respuesta como *X* y *y* respectivamente.

```
# Load the dataset
load("GroundnutToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env, G & GE
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
```



```

# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %%% Geno
LineXGenoEnv <- model.matrix(~ 0 + LineG:Env)

# Predictor and Response Variables
X <- cbind(Env, LineG, LineXGenoEnv)
y <- BurStMisc::ntile(PhenoToy$YPH, 3, result = "factor")

# First 30 responses
print(y[1:30])
#> [1] 1 2 2 1 1 1 1 3 1 1 2 3 1 3 2 2 2 3 2 3 1 3 1 1 2 1 1 1 2
#> [30] 2
#> Levels: 1 < 2 < 3

```

Tenga en cuenta que la variable de respuesta `y` es un factor con tres niveles (o categorías), lo cual es importante para que el modelo se entrene automáticamente para una variable categórica. Por esta razón, es importante factorizar esas variables de respuesta binarias o categóricas antes de usar la función `support_vector_machine`.

Posteriormente, realizamos cinco particiones aleatorias del conjunto de líneas, con 80% este conjunto para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función `cv_random` (con los parámetros por defecto). Además, creamos los data frames vacíos `Predictions` e `Hyperparams` que servirán para guardar los valores observados y predichos en cada ambiente y posteriormente evaluar la capacidad predictiva del modelo con la función `gs_summaries`.

```

# Set seed for reproducible results
set.seed(2022)
# Unique Lines
GIDs <- unique(PhenoToy$Line)
folds <- cv_random(length(GIDs))

# A data frame that will contain the variables:
Predictions <- data.frame()
Hyperparams <- data.frame()

```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta, identificando primero las líneas correspondientes a este conjunto;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace proponiendo "sigoide" como tipo de núcleo, valores entre 0 y 10 para el hiperparámetro *gamma* y valores entre 5 y 500 para el hiperparámetro de contracción, con "Optimización bayesiana" como tipo de tuneo. Cabe señalar que estos no son los únicos hiperparámetros ajustables en el modelo;

3. Con el modelo obtenido en (2) se predice la variable respuesta y en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed*, *Predicted*, 1, 2 y 3 para cada elemento del conjunto de prueba. Tenga en cuenta que, a diferencia de los ejemplos anteriores, ahora tenemos tres columnas adicionales correspondientes a las probabilidades asociadas con cada elemento correspondiente a esa categoría.
 - b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*. Con esto, *Predictions* se formatea para que sea un argumento de la función *gs_summaries*.
5. Identificación de hiperparámetros:
 - a. *HyperparamsFold* que contiene las columnas *gamma*, *coef0*, *accuracy* y *Fold*, donde *accuracy* es la precisión del modelo para cada combinación de los hiperparámetros especificados y el número de pliegue.
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
6. Se muestran los hiperparámetros óptimos.

```
# Model training and predictions of the ith partition
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")

  # Identify the training and testing Line sets
  fold <- folds[[i]]
  Lines_sam_i <- GIDs[fold$training]
  fold_i <- which(PhenoToy$Line %in% Lines_sam_i)

  # Identify the training and testing sets
  X_training <- X[fold_i, ]
  X_testing <- X[-fold_i, ]
  y_training <- y[fold_i]
  y_testing <- y[-fold_i]

  # Model training
  model <- support_vector_machine(
    x = X_training,
    y = y_training,
    kernel = "sigmoid",

    # Specify the hyperparameters
    gamma = list(min = 0, max = 10),
    coef0 = list(min = 5, max = 500),
```

```

tune_type = "Bayesian_optimization",
tune_bayes_samples_number = 5,
tune_bayes_iterations_number = 5,

# In this example the iterations wont bw shown
verbose = FALSE
)

# Prediction of the test set
predictions <- predict(model, X_testing)

# Predictions for the Fold Fold
FoldPredictions <- cbind(
  data.frame(
    Fold = i,
    Line = PhenoToy$Line[-fold_i],
    Env = PhenoToy$Env[-fold_i],
    Observed = y_testing,
    Predicted = predictions$predicted
  ),
  predictions$probabilities
)

Predictions <- rbind(Predictions, FoldPredictions)

# Hyperparams
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)

# Best hyperparams of the model
cat("*** Optimal hyperparameters: ***\n")
print(model$best_hyperparams)
}
#> *** Fold: 1 ***
#> *** Optimal hyperparameters: ***
#> $gamma
#> [1] 3.563402
#>
#> $coef0
#> [1] 350.3854
#>
#> $accuracy
#> [1] 0.3447368
#>
#> *** Fold: 2 ***
#> $gamma
#> [1] 4.056876
#>

```

```

#> $coef0
#> [1] 500
#>
#> $accuracy
#> [1] 0.3552632
#>
#> *** FoLd: 3 ***
#> *** Optimal hyperparameters: ***
#> $gamma
#> [1] 9.084241
#>
#> $coef0
#> [1] 24.92167
#>
#> $accuracy
#> [1] 0.3547368
#>
#> *** FoLd: 4 ***
#> $coef0
#> [1] 264.1667
#>
#> $accuracy
#> [1] 0.3647368
#>
#> *** FoLd: 5 ***
#> $gamma
#> [1] 8.518631
#>
#> $coef0
#> [1] 39.7706
#>
#> $accuracy
#> [1] 0.3857895

```

Predictions contiene las columnas *Fold*, *Line*, *Env*, *Observed*, *Predicted*, 1, 2* y 3 para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo los hiperparámetros óptimos (entre los posibles valores de las combinaciones de estos) que minimizan la función de coste (*pcic*: Proporción de Casos Incorrectamente Clasificados) con el tipo de sintonía “Optimización Bayesiana”, correspondiente al formato necesario para utilizar la función *gs_summaries en Predicción* en el caso de variables categóricas.

La función *gs_summaries* devuelve una lista que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```

head(Predictions)
#>   Fold   Line      Env Observed Predicted      1
#> 13    1  DTG15 ALIYARNAGAR_R15      1      3 0.3317536
#> 14    1  DTG15 ICRISAT_PR15-16      3      3 0.3317536
#> 15    1  DTG15      ICRISAT_R15      2      3 0.3317536

```

```

#> 16      1   DTG15      JALGOAN_R15      2      3 0.3317536
#> 41      1 ICG3746 ALIYARNAGAR_R15      1      3 0.3317536
#> 42      1 ICG3746 ICRISAT_PR15-16      3      3 0.3317536
#>          2          3
#> 13 0.2990924 0.369154
#> 14 0.2990924 0.369154
#> 15 0.2990924 0.369154
#> 16 0.2990924 0.369154
#> 41 0.2990924 0.369154
#> 42 0.2990924 0.369154
unique(Predictions$Fold)
#> [1] 1 2 3 4 5
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted      X1      X2      X3
#> 1 49x37-134          1          1 0.3962 0.2979 0.3059
#> 2 49x37-99(b)tall      1          1 0.3962 0.2979 0.3059
#> 3      DTG15          2          3 0.3318 0.2991 0.3692
#> 4      DTG3          2          1 0.3962 0.2979 0.3059
#> 5      Gangapuri        1          1 0.3530 0.3241 0.3229
#> 6      ICG10036         1          1 0.3571 0.3302 0.3127

# Summaries by Environment
summaries$env[, 1:5]
#>      Env      PCCC PCCC_SE      Kappa Kappa_SE
#> 1 ALIYARNAGAR_R15 0.2667 0.0850 -0.1077 0.0754
#> 2 ICRISAT_PR15-16 0.3667 0.1106 -0.0267 0.0859
#> 3      ICRISAT_R15 0.4333 0.0408 0.1282 0.0786
#> 4      JALGOAN_R15 0.3000 0.0333 -0.0573 0.0780
#> 5      Global      0.5000 0.0527 0.1440 0.1017
summaries$env[, 6:7]
#>      BrierScore BrierScore_SE
#> 1      0.6863      0.0172
#> 2      0.7025      0.0354
#> 3      0.6702      0.0115
#> 4      0.6715      0.0068
#> 5      0.6607      0.0125

# Summaries by Fold
summaries$fold
#>      Fold      PCCC PCCC_SE      Kappa Kappa_SE BrierScore
#> 1      1 0.4167 0.0833 0.0000 0.0000 0.6656
#> 2      2 0.4167 0.0481 0.0000 0.0000 0.6565
#> 3      3 0.2917 0.0798 0.0040 0.1599 0.7000
#> 4      4 0.3750 0.0417 0.0000 0.0000 0.6644

```

```
#> 5      5 0.2083  0.1250 -0.0833   0.1663   0.7268
#> 6 Global 0.5000  0.0527  0.1440   0.1017   0.6607
#>   BrierScore_SE
#> 1      0.0117
#> 2      0.0106
#> 3      0.0156
#> 4      0.0083
#> 5      0.0376
#> 6      0.0125
```

Además, Hyperparas contiene columnas *gamma*, *coef0*, *accuracy* y *Fold*, donde el valor de la columna de *precisión* corresponde a la precisión del modelo para cada combinación de hiperparámetro y valores de partición, ordenados de menor a mayor dentro de cada partición.

```
# First rows of Hyperparams
head(Hyperparams)
# Last rows of Hyperparams
tail(Hyperparams)
```

8 Deep learning

8.1 Ejemplo con respuesta continua con Grid search y particiones aleatorias solo *G* en el predictor

Este ejemplo evalúa un modelo de Red Neuronal Artificial (*Deep_learning*) con solo una capa oculta y con cinco particiones aleatorias, con 20% los datos para el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de cada partición (los parámetros predeterminados de la función *cv_random*), para un continuo respuesta, usando solo la matriz *G* (matriz de diseño de línea que contiene la información geómica) como predictor y usando "Búsqueda de cuadrícula" como tipo de ajuste para los hiperparámetros *learning_rate* y *neurons_number_1* para una sola capa oculta.

En este ejemplo, el conjunto de datos utilizado es *GroundnutToy* y el objetivo es predecir la variable continua *SYPP* del data frame *PhenoToy* utilizando la matriz *G* descrita anteriormente como predictor; por lo que identificamos las variables predictoras y de respuesta como *X* y *y* respectivamente.

```
# Load the data
load("GroundnutToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of G
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
```

```

# G matrix
LineG <- Line %*% Geno

# Predictor and Response Variables
X <- LineG
y <- PhenoToy$SYPP

# Note that y is a continuous numeric vector
class(y)
#> [1] "numeric"
typeof(y)
#> [1] "double"

```

Tenga en cuenta que la variable de respuesta *y* es una variable continua (un vector con elementos de tipo "double"), lo cual es importante para que el modelo se entrene automáticamente para este tipo de variable.

Posteriormente, realizamos cinco particiones aleatorias, con 80% los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función `cv_random` (con los parámetros por defecto). Además, creamos los data frames vacíos `Predictions` e `Hyperparams` que se utilizarán para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del modelo con la función `g_summaries`.

```

# Set seed for reproducible results
set.seed(2022)
folds <- cv_random(records_number = nrow(X))

# A data frame that will contain the variables:
Predictions <- data.frame()
Hyperparams <- data.frame()

```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace, en este ejemplo, proponiendo el valor 30 para el hiperparámetro *epochs_number*, los valores 0.01, 0.05 y 0.1 para el hiperparámetro *learning_rate*, los valores 2 y 5 para el hiperparámetro *neurons_number_1* y la función de activación softmax de la única capa oculta, con "Grid Search" como tipo de sintonía (parámetro predeterminado de *tune_type*) Cabe señalar que estos no son los únicos hiperparámetros ajustables en el modelo;
3. Con el modelo obtenido en (2) se predice la variable respuesta *SYPP* en el conjunto de prueba, con el fin de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:

- a. *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba.
 - b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*.
5. Identificación de hiperparámetros:
 - a. *HyperparamsFold* que contiene las columnas *learning_rate*, *neurons_number_1*, *mean_squared_error* y *Fold*, donde *mean_squared_error* es el costo del modelo para cada combinación de los hiperparámetros especificados y el número de *Fold*.
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
6. Se muestran los hiperparámetros óptimos del modelo obtenido en (2).

```
# Model training and predictions of the ith partition
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")
  fold <- folds[[i]]

  # Identify the training and testing sets
  X_training <- X[fold$training, ]
  X_testing <- X[fold$testing, ]
  y_training <- y[fold$training]
  y_testing <- y[fold$testing]

  # Model training
  model <- deep_learning(
    X_training,
    y_training,
    epochs_number = 30,
    learning_rate = c(0.01, 0.05, 0.1),
    layers = list(
      list(neurons_number = c(2, 5), activation = c("softmax"))
    ),
    tune_type = "grid_search",
    # In this example the iterations wont be shown
    verbose = FALSE
  )

  # Prediction of the test set
  predictions <- predict(model, X_testing)

  # Predictions for the Fold Fold
  FoldPredictions <- data.frame(
    Fold = i,
    Line = PhenoToy$Line[fold$testing],
    Env = PhenoToy$Env[fold$testing],
    Observed = y_testing,
```



```

    Predicted = predictions$predicted
  )
  Predictions <- rbind(Predictions, FoldPredictions)

  # Hyperparams
  HyperparamsFold <- model$hyperparams_grid %>%
    mutate(Fold = i)
  Hyperparams <- rbind(Hyperparams, HyperparamsFold)

  # Best hyperparams of the model
  cat("*** Optimal hyperparameters: ***\n")
  print(model$best_hyperparams)
}
#> *** Fold: 1 ***
#> *** Optimal hyperparameters: ***
#> $learning_rate
#> [1] 0.1
#>
#> $neurons_number_1
#> [1] 5
#>
#> $mean_squared_error
#> [1] 7.098522
#>
#> *** Fold: 2 ***
#> *** Optimal hyperparameters: ***
#> $learning_rate
#> [1] 0.1
#>
#> $neurons_number_1
#> [1] 5
#>
#> $mean_squared_error
#> [1] 7.252397
#>
#> *** Fold: 3 ***
#> *** Optimal hyperparameters: ***
#> $learning_rate
#> [1] 0.05
#>
#> $neurons_number_1
#> [1] 2
#>
#> $mean_squared_error
#> [1] 7.285643
#>
#> *** Fold: 4 ***
#> *** Optimal hyperparameters: ***
#> $learning_rate
#> [1] 0.1

```

```

#>
#> $neurons_number_1
#> [1] 5
#>
#> $mean_squared_error
#> [1] 8.111487
#>
#> *** Fold: 5 ***
#> *** Optimal hyperparameters: ***
#> $learning_rate
#> [1] 0.05
#>
#> $neurons_number_1
#> [1] 2
#>
#> $mean_squared_error
#> [1] 6.779659

```

Predictions data frame contiene las columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba para cada partición, donde las predicciones se realizan eligiendo los hiperparámetros óptimos (entre los posibles valores especificados de estos) que minimizan la función de costo con el tipo de tuneo "Grid Search", correspondiente al formato necesario para utilizar la función *gs_summaries* sobre estas predicciones en el caso de variables continuas.

La función *gs_summaries* devuelve una lista que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```

head(Predictions)
#>   Fold      Line      Env Observed Predicted
#> 1     1 ICGV97115  JALGOAN_R15      3.50  5.565202
#> 2     1  ICG9315   ICRISAT_R15      4.33  5.565187
#> 3     1 ICGV06099 ICRISAT_PR15-16      7.73  5.565186
#> 4     1 ICGV00248   ICRISAT_R15      6.59  5.565008
#> 5     1 ICGV05057   ICRISAT_R15     11.66  5.565206
#> 6     1 ICGV02434  JALGOAN_R15      1.60  5.565157
unique(Predictions$Fold)
#> [1] 1 2 3 4 5
# Summaries
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted Difference
#> 1 49x37-134   5.4850    5.5095    0.0245
#> 2      TG19   5.4400    5.5059    0.0659
#> 3   ICG3746   5.6400    5.5239    0.1161

```

```

#> 4 ICGV95377    5.7617    5.4997    0.2620
#> 5 ICGV97115    5.2250    5.5603    0.3353
#> 6   ICG3343    6.0300    5.6063    0.4237

# Summaries by Environment
summaries$env[, 1:7]
#>      Env      MSE MSE_SE  RMSE RMSE_SE  NRMSE
#> 1 ALIYARNAGAR_R15  3.0459 0.5347 1.7093  0.1762 1.0031
#> 2 ICRISAT_PR15-16  5.6149 1.4357 2.2290  0.4021 1.6168
#> 3   ICRISAT_R15   4.8718 1.5063 2.0601  0.3962 0.9596
#> 4   JALGOAN_R15  11.1386 2.8113 3.2089  0.4587 1.0009
#> 5      Global    5.8299 0.9512 2.3830  0.1945 1.0142
#>  NRMSE_SE
#> 1  0.0573
#> 2  0.2167
#> 3  0.0194
#> 4  0.0268
#> 5  0.0166
summaries$env[, 8:14]
#>      MAE MAE_SE      Cor Cor_SE  Intercept Intercept_SE
#> 1 1.2899 0.1682 -0.0119 0.3368 -27297.6226  40927.802
#> 2 1.9743 0.3807  0.3112 0.1657  -375.4978   6879.882
#> 3 1.7241 0.3104  0.3239 0.1577  -6590.2609   5537.950
#> 4 2.5264 0.2983 -0.2333 0.1908  20075.9040  11846.181
#> 5 1.7893 0.1108  0.0742 0.0634  -1237.2170   1400.020
#>      Slope
#> 1 4778.3831
#> 2  92.2187
#> 3 1183.1402
#> 4 -3589.7562
#> 5  223.2730
summaries$env[, 15:19]
#>      Slope_SE      R2  R2_SE  MAAPE MAAPE_SE
#> 1 7237.7188 0.4538 0.1822 0.2201  0.0479
#> 2 1222.6051 0.2067 0.1396 0.5174  0.1033
#> 3  996.1641 0.2044 0.0730 0.2975  0.0519
#> 4 2125.9219 0.2000 0.1301 0.3893  0.0442
#> 5  252.6062 0.0216 0.0158 0.3070  0.0289

# Summaries by Fold
summaries$fold[, 1:8]
#>      Fold      MSE MSE_SE  RMSE RMSE_SE  NRMSE NRMSE_SE  MAE
#> 1      1  6.9256 1.4291 2.5850  0.2848 0.9645  0.0168 2.1746
#> 2      2  6.6264 2.2634 2.4817  0.3948 1.1056  0.2010 2.0461
#> 3      3  6.1713 3.0834 2.2091  0.6560 1.2272  0.1852 1.7855
#> 4      4  3.4548 1.2827 1.7309  0.3911 1.1170  0.0993 1.3273
#> 5      5  7.6608 3.7780 2.5023  0.6829 1.3111  0.3189 2.0598
#> 6 Global  5.8299 0.9512 2.3830  0.1945 1.0142  0.0166 1.7893

```

Además, Hyperparams contiene las columnas *learning_rate*, *neurons_number_1*, *mean_squared_error* y *Fold*, donde el valor de la columna *mean_squared_error* corresponde al costo del modelo para cada combinación de hiperparámetros y partición, ordenado de menor a mayor dentro de cada partición.

```
# First rows of Hyperparams
head(Hyperparams)
#>   Learning_rate neurons_number_1 mean_squared_error FoLd
#> 6         0.10             5         7.098522      1
#> 2         0.05             2         7.199071      1
#> 5         0.05             5         7.218712      1
#> 3         0.10             2         7.226735      1
#> 1         0.01             2        16.003148      1
#> 4         0.01             5        17.515628      1

# Last rows of Hyperparams
tail(Hyperparams)
#>   Learning_rate neurons_number_1 mean_squared_error FoLd
#> 24         0.05             2         6.779659      5
#> 34         0.10             2         6.808354      5
#> 64         0.10             5         6.835389      5
#> 54         0.05             5         6.837938      5
#> 44         0.01             5        16.351594      5
#> 14         0.01             2        17.395893      5
```

8.2 Ejemplo con respuesta binaria con Grid search y validación cruzada 7-fold con *Env* + *G* el predictor

Este ejemplo evalúa un modelo de Redes Neuronales Artificiales (*Deep_learning*) con una sola capa oculta y con validación cruzada de 7 veces, para una respuesta binaria, utilizando el efecto ambiente y la matriz *G* como predictores y utilizando "Grid Search" como tipo de ajuste para los hiperparámetros *learning_rate*, *neurons_number_1* para una sola capa oculta.

En este ejemplo, el conjunto de datos utilizado es *EYTTToy* y el objetivo es predecir la variable binaria *y_bin* que es una transformación de la variable Altura de *PhenoToy*, indicando con la función *ntile* si la respuesta es mayor que la mediana de esta variable o no, utilizando la matriz de diseño de la variable *PhenoToy Env* y la matriz, *G* descrita anteriormente, como predictores; por lo que identificamos las variables predictoras de respuesta como *X* y *y* respectivamente.

```
# Load the data
load("EYTTToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of G
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
```

```

Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %%% Geno

# Predictor and Response Variables
X <- cbind(Env, LineG)
y_bin <- BurStMisc::ntile(PhenoToy$Height, 2, result = "factor")
#> Warning in BurStMisc::ntile(PhenoToy$Height, 2, result =
#> "factor"): common values across groups: 1, 2

```

Tenga en cuenta que la variable de respuesta y es una variable continua (un vector con elementos de tipo "doble"), lo cual es importante para que el modelo se entrene automáticamente para este tipo de variable.

Tenga en cuenta que la variable de respuesta y_{bin} es un factor con solo dos niveles (o categorías) lo cual es importante para que el modelo se entrene automáticamente para una variable binari. Por esta razón, es importante factorizar esas variables de respesta binarias o categóricas antes de usar la funión de *aprendizaje profundo*.

Posteriormentehacemos las particiones correspondientes a 7-Fold CV, con la ayuda de la función *cv*. Además, creamos los data frames vacíos *Predictions* e *Hyperparams* que se utilizarán para guardar los valores observados y predichos en cada ambiente y luego evaluar la capacidad predictiva del moelo con la función *gs_summaries**.

```

# Set seed for reproducible results
set.seed(2022)
folds <- cv_random(records_number = nrow(X))

# A data frame that will contain the variables:
Predictions <- data.frame()
Hyperparams <- data.frame()

```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace, en este ejemplo, proponiendo los valores 0.001, 0.01 y 0.1 para el hiperparámetro *learning_rate*, el valor 50 para el hiperparámetro *epochs_number* y los valores 2 y 5 para el hiperparámetro *neurons_number_1* de la única capa oculta, además de *relu* como una función de activación y con "Grid Search" coo tipo de sintonía (parámetro predeterminado de *tune_type*). Cabe señalar que estos no son los únicos hiperparámetros ajustabls en el modelo;

3. Con el modelo obtenido en (2), y_{bin} se predice la variable respuesta en el conjunto de prueba, con el fin de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: *Fold* number, *Line*, *Env*, *Observed*, *Predicted*, 1 y 2 para cada elemento del conjunto de prueba. Tenga en cuenta que, a diferencia del ejemplo anterior, ahora tenemos dos columnas adicionales correspondientes a las probabilidades asociadas con cada elemento correspondiente a esa categoría.
 - b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*.
5. Identificación de hiperparámetros:
 - a. *HyperparamsFold* que contiene las columnas *learning_rate*, *neurons_number_1*, *binary_crossentropy* y *Fold*, donde *binary_crossentropy* es el costo del modelo para cada combinación de los hiperparámetros especificados y el número de *Fold*.
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
6. Se muestran los hiperparámetros óptimos del modelo obtenido en (2).

```
# Model training and predictions of the ith partition
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")
  fold <- folds[[i]]

  # Identify the training and testing sets
  X_training <- X[fold$training, ]
  X_testing <- X[fold$testing, ]
  y_training <- y_bin[fold$training]
  y_testing <- y_bin[fold$testing]

  # Model training
  model <- deep_learning(
    X_training,
    y_training,
    epochs_number = 50,
    learning_rate = c(0.001, 0.01, 0.1),
    layers = list(
      list(neurons_number = c(2, 5), activation = c("relu"))
    ),
    tune_type = "grid_search",
    # In this example the iterations wont be shown
    verbose = FALSE
  )
}
```

```

# Prediction of the test set
predictions <- predict(model, X_testing)
# categorical_summary(observed = y_testing, predictions$predicted)

# Predictions for the Fold Fold
FoldPredictions <- cbind(
  data.frame(
    Fold = i,
    Line = PhenoToy$Line[fold$testing],
    Env = PhenoToy$Env[fold$testing],
    Observed = y_testing,
    Predicted = predictions$predicted
  ),
  predictions$probabilities
)
Predictions <- rbind(Predictions, FoldPredictions)

# Hyperparams
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)

# Best hyperparams of the model
cat("*** Optimal hyperparameters: ***\n")
print(model$best_hyperparams)
}
#> *** Fold: 1 ***
#> *** Optimal hyperparameters: ***
#> $learning_rate
#> [1] 0.01
#>
#> $neurons_number_1
#> [1] 2
#>
#> $binary_crossentropy
#> [1] -0.482315
#>
#> *** Fold: 2 ***
#> *** Optimal hyperparameters: ***
#> $learning_rate
#> [1] 0.01
#>
#> $neurons_number_1
#> [1] 5
#>
#> $binary_crossentropy
#> [1] -0.4674027
#>
#> *** Fold: 3 ***
#> *** Optimal hyperparameters: ***

```

```

#> $learning_rate
#> [1] 0.01
#>
#> $neurons_number_1
#> [1] 2
#>
#> $binary_crossentropy
#> [1] -0.4997773
#>
#> *** Fold: 4 ***
#> *** Optimal hyperparameters: ***
#> $learning_rate
#> [1] 0.01
#>
#> $neurons_number_1
#> [1] 2
#>
#> $binary_crossentropy
#> [1] -0.476533
#>
#> *** Fold: 5 ***
#> *** Optimal hyperparameters: ***
#> $learning_rate
#> [1] 0.01
#>
#> $neurons_number_1
#> [1] 2
#>
#> $binary_crossentropy
#> [1] -0.593723

```

Predictions contiene las columnas *Fold*, *Line*, *Env*, *Observed*, *Predicted*, *1* y *2* para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo los hiperparámetros óptimos (entre los posibles valores especificados de estos que minimizar la función de costo con el tipo de tuneo "Grid Search", correspondiente al formato necesario para usar la función *gs_summaries* en estas predicciones en el caso de variables continuas.

La función *gs_summaries* devuelve una lista que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```

head(Predictions)
#>   Fold      Line      Env Observed Predicted      1
#> 1     1 GID7632666 FlatDrip      1        1 0.67989001
#> 2     1 GID7628158 Flat5IR      1        2 0.04963881
#> 3     1 GID7631195      EHT      2        2 0.04056382
#> 4     1 GID7628467 Flat5IR      1        1 0.71969324
#> 5     1 GID7630553 Flat5IR      2        2 0.08742714
#> 6     1 GID7629600 FlatDrip      1        1 0.66124040
#>                2

```



```

#> 1 0.3201100
#> 2 0.9503612
#> 3 0.9594362
#> 4 0.2803068
#> 5 0.9125729
#> 6 0.3387596
unique(Predictions$Fold)
#> [1] 1 2 3 4 5
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted      X1      X2
#> 1 GID7462121          2          2 0.1409 0.8591
#> 2 GID7625106          1          1 0.7154 0.2846
#> 3 GID7625276          1          1 0.8970 0.1030
#> 4 GID7625985          1          1 0.4594 0.5406
#> 5 GID7626366          1          1 0.7160 0.2840
#> 6 GID7626381          1          1 0.9048 0.0952
# Summaries by Environment
summaries$env
#>      Env   PCCC PCCC_SE   Kappa Kappa_SE BrierScore
#> 1  Bed5IR 0.8143 0.0585 0.1679 0.1514 0.2767
#> 2   EHT 0.7000 0.1358 0.4038 0.1371 0.4561
#> 3 Flat5IR 0.5876 0.0269 0.0799 0.0944 0.4921
#> 4 FlatDrip 1.0000 0.0000   NaN    NA 0.0447
#> 5  Global 0.8106 0.0594 0.6136 0.1237 0.3092
#>      BrierScore_SE
#> 1      0.0827
#> 2      0.1146
#> 3      0.0344
#> 4      0.0311
#> 5      0.0619

# Summaries by Fold
summaries$fold
#>      Fold   PCCC PCCC_SE   Kappa Kappa_SE BrierScore
#> 1      1 0.7560 0.1093 0.1717 0.0834 0.4113
#> 2      2 0.8750 0.0798 0.4242 0.1714 0.1681
#> 3      3 0.6512 0.1744 0.1905 0.1650 0.3930
#> 4      4 0.7667 0.0882 0.2337 0.2051 0.3176
#> 5      5 0.8286 0.1017 0.0250 0.1945 0.2971
#> 6 Global 0.8106 0.0594 0.6136 0.1237 0.3092
#>      BrierScore_SE
#> 1      0.1396
#> 2      0.1065
#> 3      0.1567

```

```
#> 4      0.1037
#> 5      0.1035
#> 6      0.0619
```

Además, `Hyperparams` contiene las columnas `learning_rate`, `neurons_number_1`, `binary_crossentropy` y `Fold`, donde el valor de la columna `binary_crossentropy` corresponde al costo del modelo para cada combinación de hiperparámetros y partición, ordenados de menor a mayor dentro de cada partición.

```
# First rows of Hyperparams
head(Hyperparams)
#>   learning_rate neurons_number_1 binary_crossentropy Fold
#> 2      0.010           2      -0.4823150      1
#> 5      0.010           5      -0.5182668      1
#> 4      0.001           5      -0.6342605      1
#> 1      0.001           2      -0.6730702      1
#> 3      0.100           2      -1.3157409      1
#> 6      0.100           5      -1.5096076      1
# Last rows of Hyperparams
tail(Hyperparams)
#>   learning_rate neurons_number_1 binary_crossentropy Fold
#> 24      0.010           2      -0.5937230      5
#> 54      0.010           5      -0.5960173      5
#> 44      0.001           5      -0.6759355      5
#> 14      0.001           2      -0.6962166      5
#> 34      0.100           2      -1.3796079      5
#> 64      0.100           5      -1.5830518      5
```

8.3 Ejemplo con variable respuesta categórica con optimización bayesiana con línea de partición aleatoria *Env* + *G* + *GE* dentro del predictor

Este ejemplo evalúa un modelo de Red Neuronal artificial (*Deep_learning*) con dos capas ocultas y con cinco particiones aleatorias del conjunto de líneas, con 80% las líneas para el conjunto de entrenamiento y 20% para el conjunto de validación dentro de cada partición (los parámetros predeterminados de la función `cv_random`), para una respuesta categórica, usando el efecto *Environment*, la matriz *G* y la interacción entre estos dos como predictores, además de usar "Optimización Bayesiana" como tipo de ajuste para los hiperparámetros.

En este ejemplo, el dataset utilizado es *ChickpeaToy* y buscamos predecir la variable categórica, que es una transformación de la variable *DaystoMaturity* de *Phenooy*, utilizando como predictores la matriz de diseño de la variable *Env* de *PhenoToy* y la matriz *G*, descrita anteriormente; por lo que identificamos las variables predictoras y de respuesta como *X* y *y* respectivamente.

```
# Load the data
load("ChickpeaToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
```

```

#>   GenoToy

# Data preparation of G
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %%% Geno
LineGenoxEnv <- model.matrix(~ 0 + LineG:Env)

# Predictor and Response Variables
X <- cbind(Env, LineG, LineGenoxEnv)
y <- BurStMisc::ntile(PhenoToy$DaystoMaturity, 3, result = "factor")
#> Warning in BurStMisc::ntile(PhenoToy$DaystoMaturity, 3, result
#> = "factor"): common values across groups: 1, 2

# First 30 responses
print(y[1:30])
#> [1] 3 3 1 1 3 1 2 3 1 1 3 1 2 3 2 1 3 1 2 3 2 2 2 2 3 3 2 2 3
#> [30] 2
#> Levels: 1 < 2 < 3

```

Tenga en cuenta que la variable de respuesta `y` es un factor con tres niveles (o categorías), lo cual es importante para que el modelo se entrene automáticamente para un variable categórica. Por esta razón, es importante factorizar esas variables de respuesta binarias o categóricas antes de usar la función de *aprendizaje profundo*.

Posteriormente, realizamos cinco particiones aleatorias del conjunto de líneas, con 8% este conjunto para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función `cv_random` (con los parámetros por defecto). Además, creamos los data frames vacíos `Predictions` e `Hyperparams` que servirán para guardar los valores observados y predichos en cada ambiente y posteriormente evaluar la capacidad predictiva del modelo con la función `gs_summaries`.

```

# Random Partition Line
set.seed(2022)
# Unique Lines
GIDs <- unique(PhenoToy$Line)
folds <- cv_random(length(GIDs))

# A data frame that will contain the variables:
Predictions <- data.frame()
Hyperparams <- data.frame()

```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;

2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace, en este ejemplo, proponiendo dos capas ocultas, el valor 50 para el *hiperparámetro epochs_number*, el valor 0.01 para el *hiperparámetro learning_rate*, valores entre 2 y 5 para el *hiperparámetro neurons_number_1* de la primera capa oculta y valores entre 2 y 10 para el *hiperparámetro neurons_number_2*, además de *linel* como función de activación y con "Optimización bayesiana" como tipo de *uneo*. Cabe señalar que estos no son los únicos hiperparámetros ajustables en el modelo;
3. Con el modelo obtenido en (2), se predice la variable respuesta y en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed*, *Predicted*, 1, 2 y 3 para cada elemento del conjunto de prueba. Tenga en cuenta que, a diferencia del ejemplo anterior, ahora tenemos tres columnas adicionales correspondientes a las probabilidades asociadas con cada elemento que cae en esa categoría.
 - b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*.
5. Identificación de hiperparámetros:
 - a. *HyperparamsFold* que contiene las columnas *neurons_number_1*, *neurons_number_2*, *binary_crossentropy* y *Fold*, donde *binary_crossentropy* es el costo del modelo para cada combinación de los hiperparámetros especificados y el número de *Fold*.
 - b. Cada fila de *HyperparamsFold* se agrega al data frame de *Hyperparams*.
6. Se muestran los hiperparámetros óptimos del modelo obtenido en (2).

```
# Model training and predictions of the ith partition
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")

  # Identify the training and testing Line sets
  fold <- folds[[i]]
  Lines_sam_i <- GIDs[fold$training]
  fold_i <- which(PhenoToy$Line %in% Lines_sam_i)

  # Identify the training and testing sets
  X_training <- X[fold_i, ]
  X_testing <- X[-fold_i, ]
  y_training <- y[fold_i]
  y_testing <- y[-fold_i]

  # Model training
```

```

model <- deep_learning(
  X_training,
  y_training,
  epochs_number = 50,
  learning_rate = 0.01,
  layers = list(
    list(
      neurons_number = list(min = 2, max = 5),
      activation = c("linear")
    ),
    list(
      neurons_number = list(min = 2, max = 10),
      activation = c("linear")
    )
  ),
  tune_type = "Bayesian_Optimization",
  tune_bayes_iterations_number = 5,
  tune_bayes_samples_number = 5,
  # In this example the iterations wont be shown
  verbose = FALSE
)

# Prediction of the test set
predictions <- predict(model, X_testing)
categorical_summary(
  observed = y_testing,
  predicted = predictions$predicted
)

# Predictions for the Fold Fold
FoldPredictions <- cbind(
  data.frame(
    Fold = i,
    Line = PhenoToy$Line[-fold_i],
    Env = PhenoToy$Env[-fold_i],
    Observed = y_testing,
    Predicted = predictions$predicted
  ),
  predictions$probabilities
)
Predictions <- rbind(Predictions, FoldPredictions)

# Hyperparams
HyperparamsFold <- model$hyperparams_grid %>%
  mutate(Fold = i)
Hyperparams <- rbind(Hyperparams, HyperparamsFold)

# Best hyperparams of the model
cat("*** Optimal hyperparameters: ***\n")

```

```

print(model$best_hyperparams)
}
#> *** Fold: 1 ***
#> *** Optimal hyperparameters: ***
#> $neurons_number_1
#> [1] 3
#>
#> $neurons_number_2
#> [1] 9
#>
#> $categorical_crossentropy
#> [1] -0.6319248
#>
#> *** Fold: 2 ***
#> *** Optimal hyperparameters: ***
#> $neurons_number_1
#> [1] 4
#>
#> $neurons_number_2
#> [1] 9
#>
#> $categorical_crossentropy
#> [1] -0.5986446
#>
#> *** Fold: 3 ***
#> *** Optimal hyperparameters: ***
#> $neurons_number_1
#> [1] 3
#>
#> $neurons_number_2
#> [1] 4
#>
#> $categorical_crossentropy
#> [1] -0.5448596
#>
#> *** Fold: 4 ***
#> *** Optimal hyperparameters: ***
#> $neurons_number_1
#> [1] 2
#>
#> $neurons_number_2
#> [1] 7
#>
#> $categorical_crossentropy
#> [1] -0.4800281
#>
#> *** Fold: 5 ***
#> *** Optimal hyperparameters: ***
#> $neurons_number_1
#> [1] 2

```

```
#>
#> $neurons_number_2
#> [1] 2
#>
#> $categorical_crossentropy
#> [1] -0.6996199
```

Predictions contiene las columnas *Fold*, *Line*, *Env*, *Observed*, *Predicted*, 1 y 2 para cada elemento del conjunto de prueba de cada partición, donde las predicciones se realizan eligiendo los hiperparámetros óptimos (entre los posibles valores especificados de estos) que minimizar la función de costo con el tipo de tuneo "Bayesian_Optimization", correspondiente al formato necesario para usar la función *gs_summaries* e estas predicciones en el caso de variables continuas.

La función *gs_summaries* devuelve una lista que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```
head(Predictions[, 1:5])
#>   Fold      Line Env Observed Predicted
#> 1    1 ICCV03104  1         2         2
#> 2    1 ICCV03104  2         3         3
#> 3    1 ICCV03104  4         2         2
#> 4    1 ICCV03104  5         2         2
#> 5    1 ICCV03104  6         2         3
#> 6    1 ICCV03104  7         2         2
head(Predictions[, 6:8])
#>           1           2           3
#> 1 6.774122e-06 5.502464e-01 4.497468e-01
#> 2 2.029068e-07 6.309589e-09 9.999998e-01
#> 3 1.959284e-02 9.804025e-01 4.650263e-06
#> 4 1.075169e-01 8.924811e-01 1.955627e-06
#> 5 2.406484e-05 1.196870e-01 8.802890e-01
#> 6 5.515504e-02 9.448404e-01 4.470051e-06
unique(Predictions$Fold)
#> [1] 1 2 3 4 5
# Summaries
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted      X1      X2      X3
#> 1 ICCV00402         1         1 0.3003 0.3738 0.3260
#> 2 ICCV01301         1         1 0.2741 0.4656 0.2603
#> 3 ICCV03104         2         2 0.0304 0.5813 0.3883
#> 4 ICCV03105         2         2 0.1490 0.5280 0.3229
#> 5 ICCV03107         1         1 0.2859 0.3856 0.3285
#> 6 ICCV03109         1         2 0.1919 0.5655 0.2426
```

```

# Summaries by Environment
summaries$env
#>      Env  PCCC PCCC_SE  Kappa Kappa_SE BrierScore
#> 1      1 0.6333 0.0624 -0.0571 0.0571 0.6714
#> 2      2 1.0000 0.0000      NaN      NA 0.0000
#> 3      4 0.6333 0.0972 0.2167 0.2198 0.5309
#> 4      5 0.6667 0.0745 0.0864 0.0540 0.5849
#> 5      6 0.6667 0.0913 -0.1214 0.0646 0.5210
#> 6      7 0.6667 0.0000 0.1267 0.0990 0.6293
#> 7 Global 0.5667 0.0408 0.0362 0.1299 0.6923
#>      BrierScore_SE
#> 1      0.0780
#> 2      0.0000
#> 3      0.1811
#> 4      0.1007
#> 5      0.1671
#> 6      0.0331
#> 7      0.0408

# Summaries by Fold
summaries$fold
#>      Fold  PCCC PCCC_SE  Kappa Kappa_SE BrierScore
#> 1      1 0.7500 0.0714 0.2500 0.1128 0.4640
#> 2      2 0.7500 0.0938 -0.0714 0.0583 0.4317
#> 3      3 0.6667 0.0745 0.0792 0.0932 0.5540
#> 4      4 0.6111 0.0930 -0.0900 0.1140 0.6679
#> 5      5 0.7778 0.0556 0.0933 0.1355 0.3304
#> 6 Global 0.5667 0.0408 0.0362 0.1299 0.6923
#>      BrierScore_SE
#> 1      0.1254
#> 2      0.1460
#> 3      0.1322
#> 4      0.1582
#> 5      0.1006
#> 6      0.0408

```

Además, Hyperparams contiene las columnas *neurons_number_1*, *neurons_number_3*, *binary_crossentropy* y *Fold*, donde el valor de la columna *binary_crossentropy* corresponde al costo del modelo para cada combinación de hiperparámetros y partición, ordenados de menor a mayor dentro de cada partición.

```

# First rows of Hyperparams
head(Hyperparams)
#>      neurons_number_1 neurons_number_2 categorical_crossentropy
#> 4                      3                      9          -0.6319248
#> 3                      4                      5          -0.6442768
#> 7                      4                      3          -0.6451322
#> 1                      3                      3          -0.6565138
#> 8                      3                     10          -0.6714917

```



```

#> 10          2          3          -0.6775789
#>      Fold
#> 4      1
#> 3      1
#> 7      1
#> 1      1
#> 8      1
#> 10     1

# Last rows of Hyperparams
tail(Hyperparams)
#>      neurons_number_1 neurons_number_2 categorical_crossentropy
#> 14          4          6          -0.7696693
#> 84          2          7          -0.7749689
#> 24          4          3          -0.8060311
#> 64          2         10          -0.8149446
#> 74          3          9          -0.8444456
#> 44          4          6          -0.9248140
#>      Fold
#> 14      5
#> 84      5
#> 24      5
#> 64      5
#> 74      5
#> 44      5

```

9 Partial Least Squares Regression

9.1 Ejemplo con respuesta continua solo G en el predictor con Grid search y particiones aleatorias

Este ejemplo evalúa un modelo de regresión de mínimos cuadrados parciales (PLS) con cinco particiones aleatorias, con 20% los datos para el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de cada partición (los parámetros predeterminados de la función `cv_random`), para una respuesta continua, usando solo el matrix G (matriz de diseño de línea que contiene información genómica) como predictor.

En este ejemplo, el conjunto de dato utilizado es *EYTTToy* y el objetivo es predecir la variable continua *DTHD* del data frame *PhenoToy* utilizando la matriz G descrita anteriormente como predictor; por lo que identificamos las variables predictoras y de respuesta como X y y respectivamente.

```

# Load the data
load("EYTTToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

```

```

# Data preparation of G
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %**% Geno

# Predictor and Response Variables
X <- LineG
y <- PhenoToy$DTHD

# Note that y is a continuous numeric vector
class(y)
#> [1] "numeric"
typeof(y)
#> [1] "double"

```

Tenga en cuenta que la variable de respuesta *y* es una variable continua (un vector con elementos de tipo "doble"), lo cual es importante para que el modelo se entrene automáticamente para una variable continua.

Posteriormente, realizamos cinco particiones aleatorias, con 80% los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función *cv_random* (con los parámetros por defecto). Además, creamos los data frames vacíos *Predictions* e *Hyperparams* que servirán para guardar los valores observados y predichos en cada ambiente posteriormente evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```

# Set seed for reproducible results
set.seed(2022)
folds <- cv_random(records_number = nrow(X))

# A data frame that will contain the variables:
Predictions <- data.frame()
Hyperparams <- data.frame()

```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace con la *función parcial_mínimos_cuadrados*;
3. Con el modelo obtenido en (2), se predice la variable respuesta *DTHD* en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:

- a. *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* paracada elemento del conjunto de prueba.
- b. Cada fila de *FoldPrediction* se agrega al data frame *Predictions*.

```
# Model training and predictions of the ith partition
for (i in seq_along(folds)) {
  cat("\t*** Fold:", i, "****\n")
  fold <- folds[[i]]

  # Identify the training and testing sets
  X_training <- X[fold$training, ]
  X_testing <- X[fold$testing, ]
  y_training <- y[fold$training]
  y_testing <- y[fold$testing]

  # Model training
  model <- partial_least_squares(
    x = X_training,
    y = y_training,
    method = "kernel"
  )

  # Prediction of the test set
  predictions <- predict(model, X_testing)

  # Predictions for the Fold Fold
  FoldPredictions <- data.frame(
    Fold = i,
    Line = PhenoToy$Line[fold$testing],
    Env = PhenoToy$Env[fold$testing],
    Observed = y_testing,
    Predicted = predictions$predicted
  )
  Predictions <- rbind(Predictions, FoldPredictions)
}

#> *** Fold: 1 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.0414 secs ***
#> *** Fold: 2 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.0289 secs ***
#> *** Fold: 3 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.0283 secs ***
#> *** Fold: 4 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.0273 secs ***
#> *** Fold: 5 ***
```

```
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.0397 secs ***
```

Predictions data frame contiene las columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición, correspondiente al formato necesario para usar la función *gs_summaries* en estas predicciones en el caso de variables continuas.

La función *gs_summaries* devuelve una lista que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```
head(Predictions)
#>   Fold      Line      Env Observed Predicted
#> 1     1 GID7632666 FlatDrip      73  75.31904
#> 2     1 GID7628158 Flat5IR      80  81.59250
#> 3     1 GID7631195      EHT      73  76.21725
#> 4     1 GID7628467 Flat5IR      74  68.88331
#> 5     1 GID7630553 Flat5IR      74  70.01089
#> 6     1 GID7629600 FlatDrip      82  81.60776
unique(Predictions$Fold)
#> [1] 1 2 3 4 5
# Summaries
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted Difference
#> 1 GID7631195  75.0000   75.0375    0.0375
#> 2 GID7626446  74.6000   74.7329    0.1329
#> 3 GID7629552  71.0000   70.7531    0.2469
#> 4 GID7628158  81.3333   80.9967    0.3366
#> 5 GID7625985  75.0000   74.6089    0.3911
#> 6 GID7730251  74.3333   73.9141    0.4193

# Summaries by Environment
summaries$env[, 1:9]
#>      Env      MSE MSE_SE  RMSE RMSE_SE  NRMSE NRMSE_SE
#> 1 Bed5IR 14.3736 2.5695 3.7358 0.3230 0.8324 0.1780
#> 2 EHT 25.3654 3.3653 4.9958 0.3193 1.0298 0.2750
#> 3 Flat5IR 14.3807 2.8729 3.7283 0.3465 1.7541 0.2579
#> 4 FlatDrip 10.8040 2.9134 3.1528 0.4648 1.1055 0.2487
#> 5 Global 13.7911 1.6105 3.6901 0.2086 0.7811 0.0432
#>      MAE MAE_SE
#> 1 3.0321 0.1827
#> 2 3.8953 0.3293
#> 3 3.2366 0.3273
#> 4 2.7587 0.4862
```

```
#> 5 3.0087 0.1915
summaries$env[, 10:17]
#>      Cor Cor_SE Intercept Intercept_SE Slope Slope_SE      R2
#> 1 0.4185 0.3542  23.9180    45.1432 0.6761  0.6145 0.6769
#> 2 0.5437 0.2381 -22.3696    44.2738 1.2733  0.6036 0.5223
#> 3 0.6141 0.1264  30.4407    12.8880 0.6321  0.1758 0.4410
#> 4 0.4033 0.2567  35.9892    20.1051 0.5109  0.2682 0.4262
#> 5 0.6314 0.0451  -0.0998    11.6124 1.0057  0.1610 0.4069
#>      R2_SE
#> 1 0.1009
#> 2 0.1711
#> 3 0.1498
#> 4 0.1200
#> 5 0.0575
summaries$env[, 18:19]
#>      MAAPE MAAPE_SE
#> 1 0.0407  0.0021
#> 2 0.0560  0.0050
#> 3 0.0418  0.0040
#> 4 0.0376  0.0066
#> 5 0.0411  0.0029

# Summaries by Fold
summaries$fold[, 1:8]
#>      Fold      MSE MSE_SE  RMSE RMSE_SE  NRMSE NRMSE_SE  MAE
#> 1      1 12.3975 4.1486 3.3420  0.6399 1.0754  0.3288 2.7727
#> 2      2 23.2356 5.5527 4.7137  0.5822 1.4584  0.4432 3.9356
#> 3      3 13.8472 4.2969 3.6004  0.5429 1.0547  0.2432 2.7702
#> 4      4 18.3978 3.2640 4.2373  0.3845 1.2358  0.2596 3.4580
#> 5      5 13.2764 1.7029 3.6225  0.2266 1.0778  0.3491 3.2169
#> 6 Global 13.7911 1.6105 3.6901  0.2086 0.7811  0.0432 3.0087
```

9.2 Ejemplo de datos de recuento con optimización bayesiana con línea de partición aleatoria *Env* + *G* + *GE* en el predictor.

Este ejemplo evalúa un modelo de Regresión de Mínimos Cuadrados Parciales on cinco particiones aleatorias del conjunto de líneas, con 20% las líneas para el conjunto de prueba y 80% para el conjunto de entreamiento dentro de cada partición, para una respuesta de conteo, usando el efecto ambiente, la matriz *G* y la interacción entre estos dos como predictores.

En este ejemplo, el dataset utilizado es *MaizeToy* y busca predecir la variable de coneo numérico *PH*, utilizando como predictores la matriz de diseo de la variable *Env* de PhenoToy, la matriz *G* descrita anteriormente y la atriz de diseño de la interacción entre estas dos; por lo que identificamos las variables predictoras y de respuesta como *X* y *y* respectivamente.

```
# Load the dataset
load("MaizeToy.RData", verbose = TRUE)
```

```

#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env, G & GE
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
LineG <- Line %**% Geno
LinexGenoEnv <- model.matrix(~ 0 + LineG:Env)

# Predictor and Response Variables
X <- cbind(Env, LineG, LinexGenoEnv)
y <- PhenoToy$PH
print(y[1:15])
#> [1] 239 223 223 239 213 221 237 152 195 252 208 240 239 215
#> [15] 252
typeof(y)
#> [1] "integer"

```

Posteriormente, realizamos cinco particiones aleatorias del conjunto de líneas, con 80% este conjunto para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función `cv_random` (con los parámetros por defecto). Además, creamos los data frames vacíos `Predictions` e `Hyperparams` que servirán para guardar los valores observados y predichos en cada abiente y posteriormente evaluar la capacidad predictiva del modelo con la función `gs_summaries`.

```

# Random Partition
set.seed(2022)
# Unique Lines
GIDs <- unique(PhenoToy$Line)
folds <- cv_random(length(GIDs))

# A data frame that will contain the variables:
Predictions <- data.frame()
Hyperparams <- data.frame()

```

Posteriormente se seguirá el siguiente proceso **para cada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta, identificando primero las líneas correspondientes a cada conjunto;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace con la función `parcial_mínimos_cuadrados` ;
3. Identificación de predicciones del conjunto de pruebas:

- a. *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env Observed* y *Predicted* para cada elemento del conjunto de prueba.
- b. Cada fila de *FoldPredictin* se agrega al data frame *Predictions*. Con esto, *Predictions* se formatea para que sea un argumento de la función *gs_summaries*.

```
# Model training and predictions of the ith partition
for (i in seq(folds)) {
  cat("*** Fold:", i, "***\n")
  # Identify the training and testing Line sets
  fold <- folds[[i]]
  Lines_sam_i <- GIDs[fold$training]
  fold_i <- which(PhenoToy$Line %in% Lines_sam_i)

  # Identify the training and testing sets
  X_training <- X[fold_i, ]
  X_testing <- X[-fold_i, ]
  y_training <- y[fold_i]
  y_testing <- y[-fold_i]

  # Model training
  model <- partial_least_squares(
    x = X_training,
    y = y_training
  )

  # Testing Predictions
  predictions <- predict(model, X_testing)

  # Predictions for the Fold Fold
  FoldPredictions <- data.frame(
    Fold = i,
    Line = PhenoToy$Line[-fold_i],
    Env = PhenoToy$Env[-fold_i],
    Observed = y_testing,
    Predicted = predictions$predicted
  )
  Predictions <- rbind(Predictions, FoldPredictions)
}
#> *** Fold: 1 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.2988 secs ***
#> *** Fold: 2 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1563 secs ***
#> *** Fold: 3 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1344 secs ***
```

```
#> *** Fold: 4 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1323 secs ***
#> *** Fold: 5 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1538 secs ***
```

Predictions contiene columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición, correspondient al formato necesario para usar la función *gs_summaries* en *Prediction* en el caso de contar variables.

La función *gs_summaries* evuelve una lista que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```
head(Predictions)
#>   Fold      Line Env Observed Predicted
#> 1     1 CKDHL0049 EBU      252   233.4426
#> 2     1 CKDHL0049 KAK      208   202.7797
#> 3     1 CKDHL0049 KTI      240   234.5414
#> 4     1 CKDHL0108 EBU      237   233.8535
#> 5     1 CKDHL0108 KAK      219   201.8579
#> 6     1 CKDHL0108 KTI      237   235.1289
unique(Predictions$Fold)
#> [1] 1 2 3 4 5
# Summaries
summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted Difference
#> 1 CKDHL0129 224.0000  222.9664     1.0336
#> 2 CKDHL0052 220.6667  222.9475     2.2808
#> 3 CKDHL0032 224.3333  222.0165     2.3169
#> 4 CKDHL0515 220.6667  223.1351     2.4685
#> 5 CKDHL0530 220.0000  222.5652     2.5652
#> 6 CKDHL0150 220.3333  223.4488     3.1154

# Summaries by Environment
summaries$env[, 1:8]
#>      Env      MSE MSE_SE      RMSE RMSE_SE  NRMSE NRMSE_SE
#> 1    EBU  99.7003 22.6809   9.7330   1.1145 0.8853   0.0490
#> 2    KAK 197.4719 33.9231  13.8323   1.2388 0.9880   0.0422
#> 3    KTI 316.2919 67.2268  17.2618   2.1402 1.2422   0.3107
#> 4 Global 106.5268 19.6411  10.1300   0.9886 1.0033   0.0860
#>      MAE
#> 1   8.8154
#> 2  11.6273
```



```

#> 3 14.3905
#> 4 8.1770
summaries$env[, 9:16]
#>   MAE_SE   Cor Cor_SE Intercept Intercept_SE   Slope
#> 1 1.1000 0.1816 0.2292 -5718.386    5640.774 25.5061
#> 2 1.2231 0.2507 0.1974 -4495.943    4317.875 23.4379
#> 3 1.5920 0.4059 0.1984 -13066.277   10355.917 57.1038
#> 4 0.8667 0.6301 0.1559 -10563.040    6895.127 48.5133
#>   Slope_SE   R2
#> 1 24.2028 0.2431
#> 2 21.5624 0.2187
#> 3 44.5019 0.3222
#> 4 31.0693 0.4942
summaries$env[, 17:19]
#>   R2_SE MAAPE MAAPE_SE
#> 1 0.0686 0.0382 0.0048
#> 2 0.1551 0.0578 0.0058
#> 3 0.1392 0.0628 0.0068
#> 4 0.1802 0.0368 0.0040

# Summaries by Fold
summaries$fold[, 1:8]
#>   Fold      MSE   MSE_SE   RMSE RMSE_SE  NRMSE NRMSE_SE
#> 1     1 241.8583 67.7607 15.2683 2.0902 0.9377 0.0081
#> 2     2 114.8901 17.6555 10.6590 0.7988 0.9499 0.0145
#> 3     3 279.2435 122.0329 15.6084 4.2204 1.4298 0.5409
#> 4     4 178.7284 56.8237 12.9212 2.4259 0.9622 0.0516
#> 5     5 207.7198 96.7269 13.5884 3.3966 0.9130 0.0098
#> 6 Global 106.5268 19.6411 10.1300 0.9886 1.0033 0.0860
#>   MAE
#> 1 12.1483
#> 2 9.6012
#> 3 13.0718
#> 4 12.1904
#> 5 11.0437
#> 6 8.1770
summaries$fold[, 9:15]
#>   MAE_SE   Cor Cor_SE Intercept Intercept_SE   Slope
#> 1 1.0403 0.1636 0.3344 -1273.7458    2649.7007 6.2433
#> 2 1.2158 0.0253 0.1814 -1142.3374    2311.8597 5.5627
#> 3 3.4838 0.2623 0.2116   155.1282     54.9884 0.3024
#> 4 2.3603 0.7433 0.0982 -34334.5355   9988.6695 153.4747
#> 5 2.5674 0.2025 0.3306 -2205.5197    2564.8888 11.1634
#> 6 0.8667 0.6301 0.1559 -10563.0398    6895.1265 48.5133
#>   Slope_SE
#> 1 11.3576
#> 2 9.9979
#> 3 0.2417
#> 4 39.9157
#> 5 11.1460

```

```
#> 6 31.0693
summaries$fold[, 16:19]
#>      R2  R2_SE  MAAPE  MAAPE_SE
#> 1 0.2504 0.2131 0.0574 0.0059
#> 2 0.0665 0.0420 0.0430 0.0047
#> 3 0.1583 0.1537 0.0587 0.0151
#> 4 0.5718 0.1434 0.0547 0.0112
#> 5 0.2596 0.0376 0.0508 0.0114
#> 6 0.4942 0.1802 0.0368 0.0040
```

9.3 Ejemplo de multivariado con respuestas continuas con optimización bayesiana con validación cruzada 7-fold con *Env* + *G* el predictor

Este ejemplo evalúa un modelo de regresión de mínimos cuadrados parciales con validación cruzada 7-fold, para dos respuestas continuas, utilizando el efecto del ambiente y la matriz *G* como predictores.

En este ejemplo, el conjunto de datos utilizado es *GroundnutToy* y el objetivo es predecir las variables continuas *PYPP* y *SYPP* del marco de datos *PhenoToy* utilizando la matriz de diseño de la variable *PhenoToy Env* y la matriz como *G* predictores; por lo que identificamos las variables predictoras y de respuesta como *X* y *y* respectivamente.

```
# Load the data
load("GroundnutToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env & G
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LineG <- Line %*% Geno

# Predictor and Response Variables
X <- cbind(Env, LineG)
y <- PhenoToy[, c("PYPP", "SYPP")]
```

Posteriormente hacemos 7 particiones aleatorias, con la ayuda de la función *cv_kfold*. Además, creamos los data frames vacíos *PredictionsPYPP*, *PredictionsSYPP* e *Hyperparams* que servirán para guardar los valores observados y predichos en cada ambiente y posteriormente evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
# Set seed for reproducible results
set.seed(2022)
folds <- cv_kfold(records_number = nrow(X), k = 7)

# Data frames that will contain the variables:
```

```
PredictionsPYPP <- data.frame()
PredictionsSYPP <- data.frame()
Hyperparams <- data.frame()
```

Posteriormente, se seguirá l siguiente proceso **para cada partición y para cada variable de respuesta**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace con la función *parcial_mínimos_cuadrados* ;
3. Con el modelo obtenido en (2), se predice la variable respuesta en el conjunto de prueba, con el fin de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. Se crean los data frames *FoldPredictionsPYPP* y *FoldPredictionSYPP* que contienen las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Prediced* para cada elemento del conjunto de prueba y para cada variable de respuesta respectiva.
 - b. Cada fila de *FoldPredictionPYPP* se agrega al data frame *PredictionsPYPP* ; y cada fila de *FoldPredictionSYPP* se agrega al data frame *PredictionsSYPP*.

```
# Model training and predictions of the ith partition
```

```
for (i in seq_along(folds)) {
  cat("*** Fold:", i, "***\n")
  fold <- folds[[i]]

  # Identify the training and testing sets
  X_training <- X[fold$training, ]
  X_testing <- X[fold$testing, ]
  y_training <- y[fold$training, ]
  y_testing <- y[fold$testing, ]

  # Model training
  model <- partial_least_squares(
    x = X_training,
    y = y_training
  )

  # Testing Predictions
  predictions <- predict(model, X_testing)

  # Predictions of PYPP for the Fold
  PredictionsPYPP <- data.frame(
    Fold = i,
```

```

    Line = PhenoToy$Line[fold$testing],
    Env = PhenoToy$Env[fold$testing],
    Observed = y_testing$PYPP,
    Predicted = predictions$PYPP$predicted
  )
PredictionsPYPP <- rbind(PredictionsPYPP, FoldPredictionsPYPP)

# Predictions of ASI for the Fold
PredictionsSYPP <- data.frame(
  Fold = i,
  Line = PhenoToy$Line[fold$testing],
  Env = PhenoToy$Env[fold$testing],
  Observed = y_testing$SYPP,
  Predicted = predictions$SYPP$predicted
)
PredictionsSYPP <- rbind(PredictionsSYPP, FoldPredictionsSYPP)
}
#> *** Fold: 1 ***
#> *** Fitting Multivariate Partial Least Squares model ***
#> *** Model evaluation completed in 0.0488 secs ***
#> *** Fold: 2 ***
#> *** Fitting Multivariate Partial Least Squares model ***
#> *** Model evaluation completed in 0.0584 secs ***
#> *** Fold: 3 ***
#> *** Fitting Multivariate Partial Least Squares model ***
#> *** Model evaluation completed in 0.0466 secs ***
#> *** Fold: 4 ***
#> *** Fitting Multivariate Partial Least Squares model ***
#> *** Model evaluation completed in 0.0604 secs ***
#> *** Fold: 5 ***
#> *** Fitting Multivariate Partial Least Squares model ***
#> *** Model evaluation completed in 0.053 secs ***
#> *** Fold: 6 ***
#> *** Fitting Multivariate Partial Least Squares model ***
#> *** Model evaluation completed in 0.0708 secs ***
#> *** Fold: 7 ***
#> *** Fitting Multivariate Partial Least Squares model ***
#> *** Model evaluation completed in 0.0479 secs ***

```

Repitiendo este proceso para cada partición, los *data frames* *PredictionsPYPP* y *PredictionsSYPP* contienen las columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición en su respectiva variable de respuesta, correspondiente al formato necesario para usar la función *gs_summaries* en estas predicciones en el caso de variables continuas.

La función *gs_summaries* devuelve una lista que contiene tres *data frames* correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.

```

head(PredictionsPYPP)
#>   Fold   Line   Env Observed Predicted

```

```

#> 1      7 CSMG84-1 ALIYARNAGAR_R15      9.63 10.049876
#> 2      7 CSMG84-1      ICRISAT_R15     10.27  9.598937
#> 3      7      DTG15      ICRISAT_R15     10.44  9.906264
#> 4      7      DTG3      ICRISAT_R15      6.05 10.394036
#> 5      7 ICG15419 ALIYARNAGAR_R15      4.82 10.398460
#> 6      7 ICG15419      ICRISAT_R15      8.50  9.947522
unique(PredictionsPYPP$Fold)
#> [1] 7
head(PredictionsSYPP)
#>   Fold      Line      Env Observed Predicted
#> 1      7 CSMG84-1 ALIYARNAGAR_R15      4.69  5.854455
#> 2      7 CSMG84-1      ICRISAT_R15      5.16  5.589682
#> 3      7      DTG15      ICRISAT_R15      6.54  5.758318
#> 4      7      DTG3      ICRISAT_R15      3.65  6.069388
#> 5      7 ICG15419 ALIYARNAGAR_R15      2.64  6.079172
#> 6      7 ICG15419      ICRISAT_R15      4.68  5.814400
unique(PredictionsSYPP$Fold)
#> [1] 7
# Summaries
summariesPYPP <- gs_summaries(PredictionsPYPP)
summariesSYPP <- gs_summaries(PredictionsSYPP)

# Elements of summaries
names(summariesPYPP)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summariesPYPP$line)
#>      Line Observed Predicted Difference
#> 1      DTG15    10.440    10.4806    0.0406
#> 2 CSMG84-1     9.950     9.8197    0.1303
#> 3 ICGV99085     8.990     9.4941    0.5041
#> 4 ICGV00248    12.380    11.7165    0.6635
#> 5 ICG3746      8.300     9.0531    0.7531
#> 6      TG19     5.995     7.5890    1.5940

head(summariesSYPP$line)
#>      Line Observed Predicted Difference
#> 1 ICGV00248     6.590     6.8210    0.2310
#> 2 ICGV99085     5.280     5.5392    0.2592
#> 3      DTG15     6.540     6.1192    0.4208
#> 4 ICG3746     5.900     5.3567    0.5433
#> 5 CSMG84-1     4.925     5.6990    0.7740
#> 6      TG19     3.665     4.4791    0.8141

# Summaries by Environment
summariesPYPP$env[, 1:8]
#>      Env      MSE MSE_SE  RMSE RMSE_SE  NRMSE
#> 1 ALIYARNAGAR_R15  9.3859    NA 3.0636    NA 1.3441
#> 2 ICRISAT_PR15-16 12.4721    NA 3.5316    NA 3.1021
#> 3 ICRISAT_R15     8.7848    NA 2.9639    NA 0.9548

```

```

#> 4      JALGOAN_R15 29.8736      NA 5.4657      NA 0.6336
#> 5      Global 12.7716      NA 3.5737      NA 0.8661
#>  NRMSE_SE      MAE
#> 1      NA 2.3298
#> 2      NA 3.1080
#> 3      NA 2.1393
#> 4      NA 4.6935
#> 5      NA 2.7530
summariesPYPP$env[, 9:16]
#>  MAE_SE      Cor Cor_SE Intercept Intercept_SE      Slope
#> 1      NA -0.4778      NA 19.0721      NA -1.0596
#> 2      NA 1.0000      NA 2.4616      NA 0.3243
#> 3      NA 0.0176      NA 9.4291      NA 0.0654
#> 4      NA 1.0000      NA -36.2374      NA 4.3369
#> 5      NA 0.4693      NA -4.6508      NA 1.4172
#>  Slope_SE      R2
#> 1      NA 0.2283
#> 2      NA 1.0000
#> 3      NA 0.0003
#> 4      NA 1.0000
#> 5      NA 0.2202
summariesPYPP$env[, 17:19]
#>  R2_SE  MAAPE  MAAPE_SE
#> 1      NA 0.2959      NA
#> 2      NA 0.4986      NA
#> 3      NA 0.2173      NA
#> 4      NA 0.2853      NA
#> 5      NA 0.2819      NA

# Summaries by Fold
summariesPYPP$fold[, 1:9]
#>  Fold      MSE MSE_SE  RMSE RMSE_SE  NRMSE NRMSE_SE      MAE
#> 1      7 15.1291 4.9807 3.7562 0.5831 1.5086 0.5507 3.0676
#> 2 Global 12.7716      NA 3.5737      NA 0.8661      NA 2.7530
#>  MAE_SE
#> 1 0.581
#> 2 NA
summariesPYPP$fold[, 10:17]
#>  Cor Cor_SE Intercept Intercept_SE  Slope Slope_SE      R2
#> 1 0.3849 0.3692 -1.3186 12.1275 0.9167 1.1789 0.5572
#> 2 0.4693      NA -4.6508      NA 1.4172      NA 0.2202
#>  R2_SE
#> 1 0.2599
#> 2 NA
summariesPYPP$fold[, 18:19]
#>  MAAPE  MAAPE_SE
#> 1 0.3243 0.0607
#> 2 0.2819      NA

```

9.4 Ejemplo con kernels con Grid search y particiones aleatorias

Este ejemplo evalúa un modelo de regresión de mínimos cuadrados parciales con cinco particiones aleatorias, con 20% los datos para el conjunto de prueba y 0% para el conjunto de entrenamiento dentro de cada partición (los parámetros predeterminados de la función `cv_random`), para una respuesta continua, utilizando la matriz de diseño del PhenoToy Env variable, a matriz descrita G anteriormente y la matriz de diseño de la interacción entre estos dos, como predictores. Todo esto para los tipos de Kernel: "Linear", "Polinomial", "Sigmoid", "Gaussian", "Exponential", "Arc_csine" y "Arc_cosine_L".

En este ejemplo, el conjunto de datos utilizado es *MaizeToy* y busca predecir la variable continua *Bioass* del data frame *PhenoToy* utilizando la matriz de diseño de la variable PhenoToy Env, la matriz descrita G anteriormente y la matriz de diseño de la interacción entre estos dos, como predictores. ; por lo que identificamos las variables predictoras y de respuesta como X y y respectivamente.

```
# Load the dataset
load("MaizeToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env, G & GE
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line
Geno <- cholesky(GenoToy[, -1])
# G matrix
LinexGeno <- Line %*% Geno
LinexGenoEnv <- model.matrix(~ 0 + LinexGeno:Env)

# Predictor and Response Variables
X <- cbind(Env, LinexGeno, LinexGenoEnv)
y <- PhenoToy$Yield

dim(X)
#> [1] 90 123
print(y[1:7])
#> [1] 6.11 6.21 5.32 6.62 5.60 6.24 5.24
typeof(y)
#> [1] "double"
```

Tenga en cuenta que la variable de respuesta y es una variable continua (un vector con elementos de tipo "doble").

A diferencia de los ejemplos anteriores, ahora buscamos evaluar el modelo para cada tipo de kernel mencionado anteriormente. Por ello, creamos un vector en el que indicamos los tipos de kernel que queremos aplicar a la matriz X . Además, creamos las listas vacías *PredictionsAll*, *TimesAll* y *SummariesAll* que se utilizarán para guardar las predicciones, los

tiempos de ejecución y los resúmenes de cada modelo entrenado, es decir, para cada tipo de kernel; lo que a su vez servirá para guardar los valores observados y predichos en cada ambiente y así evaluar la capacidad predictiva del modelo con la función *gs_summaries*.

```
kernels <- c(
  "linear",
  "polynomial",
  "sigmoid",
  "Gaussian",
  "exponential",
  "arc_cosine",
  "Arc_cosine_L"
)

# Example: Apply the Linear Kernel to the data
X_Linear <- kernelize(X, kernel = kernels[1])

# Note that X_Linear is an square matrix
dim(X_Linear)
#> [1] 90 90

# Empty Lists that will contain Predictions,
# Times of execution & Summaries for each type of kernel
PredictionsAll <- list()
TimesAll <- list()
SummariesAll <- list()
```

Posteriormente, se seguirá el siguiente proceso **para cada tipo de Kernel**:

1. identifique la variable *arc_deep* con el valor 2. Si el tipo de Kernel es “Arc_cosine_L”, el valor de la variable *arc_deep* se cambia a 3 y el *kernel_type* se identifica como “Arc_cosine”; de lo contrario, *kernel_type* se identifica como el kernel predeterminado.
2. El tipo de kernel establecido en (1) se aplica a la matriz de datos *X*, asignando al argumento *arc_cosine_dep* el valor establecido en la variable *arc_deep*. Tenga en cuenta que el argumento *arc_cosine_deep* se ignora si el tipo de kernel no es *Arc_cosine*.
3. Luego realizamos cinco particiones aleatorias, con 80% los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función *cv_random*.
4. Predictions y *Tiempos* que se utilizarán para guardar los valores observados y predichos en cada ambiente y posteriormente evaluar la capacidad predictiva del modelo con la función *gs_summaries*, además de guardar los tiempos de ejecución de cada modelo entrenado para cada partición.
5. **Para cada partición:**
 1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;

2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace con la ayuda de la función *parcial_mínimos_cuadrados* ;
 3. Con el modelo obtenido en (2), se predice la variable respuesta y en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
 4. Identificación de las predicciones del conjunto de prueba: Se crea el data frame *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba. Además, cada fila de *FoldPrediction* se agrega al data frame *Predictions**.
 5. Identificación del tiempo de ejecución del entrenamiento del modelo obtenido en (2): Se crea el data frame *FoldTime* que contiene la columna que especifica el tipo de *Kernel* utilizado para entrenar el modelo, el número de *Fold* y la columna *Minutos que indica el* tiempo de ejecución, en minutos, del entrenamiento del modelo obtenido en (2). Además, cada fila de *FoldTime* se agrega al data frame *Times*.
- data frame de predicciones contiene columnas *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba de cada partición, correspondiente al formato necesario para usar la función *gs_summaries* en estas predicciones en el caso de variables continuas.
6. *Predictions* y con la ayuda de la función *gs_summaries*, evaluamos la capacidad predictiva del modelo entrenado para el tipo de kernel especificado. La función *gs_summaries* devuelve una lista que identificamos como *summaries* y que contiene tres data frames correspondientes a los resúmenes por *línea*, *env* (ambiente) y *fold*.
 7. Finalmente, se crea un elemento con el nombre de kernel especificado en cada una de las listas *PredictionsAll*, *TimesAll*, *HyperparamsAll* y *SummariesAll*, que corresponden a los marcos de datos de las listas *Predictions*, *Times*, *Hyperparams* y *summaries*, respectivamente.

```
for (kernel in kernels) {
  cat("*** Kernel:", kernel, "***\n")

  # Identify the arc_deep and the kernel
  arc_deep <- 2
  if (kernel == "Arc_cosine_L") {
    arc_deep <- 3
    kernel <- "arc_cosine"
  } else {
    kernel <- kernel
  }

  # Compute the kernel
  X <- kernelize(X, kernel = kernel, arc_cosine_deep = arc_deep)
```

```

# Random Partition
set.seed(2022)
folds <- cv_random(
  records_number = nrow(X),
  folds_number = 5,
  testing_proportion = 0.2
)

# Empty data frames that will contain Predictions, Times
# of execution & Summaries for each partition
Predictions <- data.frame()
Times <- data.frame()
Hyperparams <- data.frame()

for (i in seq_along(folds)) {
  cat("\t*** Fold:", i, "****\n")
  fold <- folds[[i]]

  # Identify the training and testing sets
  X_training <- X[fold$training, ]
  X_testing <- X[fold$testing, ]
  y_training <- y[fold$training]
  y_testing <- y[fold$testing]

  # Model training
  model <- partial_least_squares(
    x = X_training,
    y = y_training
  )

  # Testing Predictions
  predictions <- predict(model, X_testing)

  # Predictions for the Fold Fold
  FoldPredictions <- data.frame(
    Fold = i,
    Line = PhenoToy$Line[fold$testing],
    Env = PhenoToy$Env[fold$testing],
    Observed = y_testing,
    Predicted = predictions$predicted
  )
  Predictions <- rbind(Predictions, FoldPredictions)

  # Execution times
  FoldTime <- data.frame(
    kernel = kernel,
    Fold = i,
    Minutes = as.numeric(model$execution_time, units = "mins")
  )
}

```

```

    Times <- rbind(Times, FoldTime)
  }
  # Summaries of the Folds
  summaries <- gs_summaries(Predictions)

  # Predictions, Times of execution & Summaries for the
  # specified Kernel
  PredictionsAll[[kernel]] <- Predictions
  TimesAll[[kernel]] <- Times
  SummariesAll[[kernel]] <- summaries
}
#> *** Kernel: linear ***
#> *** Fold: 1 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1154 secs ***
#> *** Fold: 2 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1325 secs ***
#> *** Fold: 3 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1489 secs ***
#> *** Fold: 4 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1055 secs ***
#> *** Fold: 5 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1085 secs ***
#> *** Kernel: polynomial ***
#> *** Fold: 1 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1248 secs ***
#> *** Fold: 2 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1257 secs ***
#> *** Fold: 3 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1253 secs ***
#> *** Fold: 4 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.2203 secs ***
#> *** Fold: 5 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1783 secs ***
#> *** Kernel: sigmoid ***
#> *** Fold: 1 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1062 secs ***
#> *** Fold: 2 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1018 secs ***

```

```

#> *** Fold: 3 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1127 secs ***
#> *** Fold: 4 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1013 secs ***
#> *** Fold: 5 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1195 secs ***
#> *** Kernel: Gaussian ***
#> *** Fold: 1 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1014 secs ***
#> *** Fold: 2 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1119 secs ***
#> *** Fold: 3 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1077 secs ***
#> *** Fold: 4 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.117 secs ***
#> *** Fold: 5 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1023 secs ***
#> *** Kernel: exponential ***
#> *** Fold: 1 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1026 secs ***
#> *** Fold: 2 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1109 secs ***
#> *** Fold: 3 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1123 secs ***
#> *** Fold: 4 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.2494 secs ***
#> *** Fold: 5 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1015 secs ***
#> *** Kernel: arc_cosine ***
#> *** Fold: 1 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1064 secs ***
#> *** Fold: 2 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1012 secs ***
#> *** Fold: 3 ***
#> *** Fitting Partial Least Squares model ***

```

```

#> *** Model evaluation completed in 0.1005 secs ***
#> *** Fold: 4 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1008 secs ***
#> *** Fold: 5 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.103 secs ***
#> *** Kernel: Arc_cosine_L ***
#> *** Fold: 1 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.102 secs ***
#> *** Fold: 2 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1072 secs ***
#> *** Fold: 3 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1017 secs ***
#> *** Fold: 4 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1108 secs ***
#> *** Fold: 5 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.1134 secs ***

```

Recordando que este proceso se realizó para cada tipo de kernel, cada una de las listas *PredictionsAll*, *TimesAll* y *SummariesAll* contiene las predicciones, los tiempos de ejecución y los resúmenes, respectivamente, para cada tipo de kernel aplicado a la matriz de datos *X*. A modo de ejemplo, a continuación se muestran los resultados obtenidos para el tipo de kernel “Arc_cosine”:

```

# Predictions for the Linear Kernel
head(PredictionsAll$arc_cosine)
#>   Fold      Line Env Observed Predicted
#> 1     1 CKDHL0206 KTI      5.01  5.919401
#> 2     1 CKDHL0512 KTI      5.58  5.928278
#> 3     1 CKDHL0236 EBU      6.46  6.489596
#> 4     1 CKDHL0502 KTI      8.09  5.927514
#> 5     1 CKDHL0474 KAK      4.72  5.040961
#> 6     1 CKDHL0032 KTI      6.24  5.911333

# Times of execution for the Linear Kernel
TimesAll$arc_cosine
#>      kernel Fold      Minutes
#> 1 arc_cosine    1 0.001767675
#> 2 arc_cosine    2 0.001780144
#> 3 arc_cosine    3 0.001602209
#> 4 arc_cosine    4 0.001638361
#> 5 arc_cosine    5 0.001691206
# Elements of SummariesAll
names(SummariesAll)

```

```

#> [1] "linear"      "polynomial"  "sigmoid"     "Gaussian"
#> [5] "exponential" "arc_cosine"
# Elements of summaries for the Linear Kernel
names(SummariesAll$arc_cosine)
#> [1] "line" "env"  "fold"
# Summaries by Line
head(SummariesAll$arc_cosine$line)
#>      Line Observed Predicted Difference
#> 1 CKDHL0108   5.7600    5.7969    0.0369
#> 2 CKDHL0097   5.2800    5.2285    0.0515
#> 3 CKDHL0529   5.8067    5.8820    0.0754
#> 4 CKDHL0129   5.8800    5.7978    0.0822
#> 5 CKDHL0150   5.7825    5.8976    0.1151
#> 6 CKDHL0052   6.3200    6.2032    0.1168

# Summaries by Environment
SummariesAll$arc_cosine$env[, 1:8]
#>      Env    MSE MSE_SE    RMSE RMSE_SE  NRMSE NRMSE_SE    MAE
#> 1  EBU 0.4523 0.1060 0.6523 0.0818 1.2030 0.1876 0.4964
#> 2  KAK 0.5727 0.2370 0.6864 0.1594 1.0522 0.1016 0.5035
#> 3  KTI 1.1226 0.1635 1.0496 0.0721 0.9834 0.0452 0.8821
#> 4 Global 0.6182 0.0709 0.7812 0.0446 0.8885 0.0330 0.5908
SummariesAll$arc_cosine$env[, 9:15]
#>    MAE_SE    Cor Cor_SE Intercept Intercept_SE    Slope
#> 1 0.0399 0.2914 0.3716   -2.7967    11.1752 1.4087
#> 2 0.0969 -0.0419 0.1749    6.1610     5.0719 -0.2075
#> 3 0.0688 0.2921 0.1414  -133.2660    89.1909 23.4135
#> 4 0.0351 0.4125 0.0580    1.2558     0.6843 0.7944
#> Slope_SE
#> 1 1.7154
#> 2 0.9777
#> 3 15.0820
#> 4 0.1151
SummariesAll$arc_cosine$env[, 16:19]
#>      R2  R2_SE  MAAPE MAAPE_SE
#> 1 0.6372 0.1531 0.0790 0.0101
#> 2 0.1241 0.0666 0.1111 0.0245
#> 3 0.1653 0.0540 0.1484 0.0126
#> 4 0.1836 0.0474 0.1021 0.0069

# Summaries by Fold
SummariesAll$arc_cosine$fold[, 1:8]
#>      Fold    MSE MSE_SE    RMSE RMSE_SE  NRMSE NRMSE_SE    MAE
#> 1 1 1.1397 0.3108 1.0488 0.1409 1.0990 0.1429 0.8164
#> 2 2 0.6533 0.2806 0.7607 0.1933 1.0822 0.0815 0.5684
#> 3 3 0.4767 0.2547 0.6368 0.1887 0.8643 0.0362 0.5599
#> 4 4 0.5147 0.1737 0.6988 0.1149 1.1355 0.1687 0.5583
#> 5 5 0.7949 0.3353 0.8355 0.2200 1.2167 0.2973 0.6336
#> 6 Global 0.6182 0.0709 0.7812 0.0446 0.8885 0.0330 0.5908
SummariesAll$arc_cosine$fold[, 9:15]

```

```

#>   MAE_SE      Cor Cor_SE Intercept Intercept_SE   Slope
#> 1 0.1527  0.0049 0.1918 -148.7755    161.2367 26.0377
#> 2 0.1825 -0.1607 0.4285  -32.2639     50.7331  6.5063
#> 3 0.1714  0.6114 0.1216  -28.7752     19.9741  5.7148
#> 4 0.0972  0.0052 0.4260   10.8779     23.2138 -1.1536
#> 5 0.1174  0.4419 0.2409  -17.5661     11.5221  3.9194
#> 6 0.0351  0.4125 0.0580    1.2558      0.6843  0.7944
#>   Slope_SE
#> 1 27.3006
#> 2  8.4389
#> 3  3.2952
#> 4  3.7397
#> 5  1.8792
#> 6  0.1151
SummariesAll$arc_cosine$fold[, 17:19]
#>   R2_SE  MAAPE  MAAPE_SE
#> 1 0.0386 0.1557  0.0196
#> 2 0.2573 0.0926  0.0293
#> 3 0.1609 0.0962  0.0319
#> 4 0.1847 0.0946  0.0109
#> 5 0.2706 0.1250  0.0315
#> 6 0.0474 0.1021  0.0069

```

9.5 Ejemplo con Sparse Kernels con Grid search y particiones aleatorias

Este ejemplo evalúa un modelo de regresión de mínimos cuadrados parciales con cinco particiones aleatorias, con 20% los datos para el conjunto de prueba y 80% para el conjunto de entrenamiento dentro de cada partición (los parámetros predeterminados de la función `cv_random`), para una respuesta continua, utilizando la matriz de diseño del PhenoToy Env variable, la matriz descrita *G* anteriormente y la matriz de diseño de la interacción entre estos dos como predictores. Todo esto para el kernel tipo “Sparse_Arc_cosine” con las proporciones de líneas 0.8, aplicadas a las variables predictoras.

En este ejemplo, el conjunto de datos utilizado es *ChickpeaToy* y buscamos predecir la variable continua *DaystoMaturity* del marco de datos *PhenoTy* utilizando la matriz de diseño de la variable PhenoToy Env, a matriz descrita *G* anteriormente y la matriz de diseño de la interacción entre estos dos, como predictores. ; por lo que identificamos las variables predictoras y de respuesta como *X* y *y* respectivamente.

```

# Load the data
load("ChickpeaToy.RData", verbose = TRUE)
#> Loading objects:
#>   PhenoToy
#>   GenoToy

# Data preparation of Env, G & GE
Line <- model.matrix(~ 0 + Line, data = PhenoToy)
Env <- model.matrix(~ 0 + Env, data = PhenoToy)
# First column is Line

```

```

Geno <- cholesky(GenoToy[, -1])
# G matrix
LinexGeno <- Line %**% Geno
LinexGenoEnv <- model.matrix(~ 0 + LinexGeno:Env)

# Predictor and Response Variables
X <- cbind(Env, LinexGeno, LinexGenoEnv)
y <- PhenoToy$DaystoMaturity

dim(X)
#> [1] 180 216
print(y[1:7])
#> [1] 107.00000 154.00000 91.66667 90.33333 107.33333 93.33333
#> [7] 103.66667
typeof(y)
#> [1] "double"

```

Posteriormente identificamos el tipo de *kernel* con “Sparse_arc_cosine” y *line_proportion* con el valor 0.8 y realizamos cinco particiones aleatorias, con 80% los datos para el conjunto de entrenamiento y 20% para el conjunto de prueba, con la ayuda de la función `cv_random` (con parámetros por defecto). Además, creamos los data frames vacíos Predictions e Hyperparams que servirán para guardar los valores observados y predichos en cada ambiente y posteriormente evaluar la capacidad predictiva del modelo con la función `gs_summaries`.

```

kernels <- "Sparse_Arc_cosine"
Line_proportion <- 0.8

# Set seed for reproducible results
set.seed(2022)
folds <- cv_random(
  records_number = nrow(X),
  folds_number = 5,
  testing_proportion = 0.2
)

# Empty data frames that will contain Predictions, Times
# of execution & Summaries for each partition
Predictions <- data.frame()
Times <- data.frame()
Hyperparams <- data.frame()

```

El conjunto de tipos de kernel se aplica luego a la matriz de datos *X*, asignando el valor numérico 2 al argumento *arc_cosine_deep* y el valor establecido en la variable *line_proportion* al argumento *rows_proportion*.

```

# Compute the kernel
X <- kernelize(
  X,
  kernel = kernel,
  arc_cosine_deep = 2,

```



```

    rows_proportion = line_proportion
)

```

Posteriormente, se seguirá el siguiente proceso **para cada partición**:

1. Se identifican el conjunto de entrenamiento y el conjunto de prueba de las variables predictoras y de respuesta;
2. El modelo se entrena con el conjunto de entrenamiento. Esto se hace con la función *parcial_mínimos_cuadrados*;
3. Con el modelo obtenido en (2), se predice la variable respuesta *DaystoMaturity* en el conjunto de prueba, con el objetivo de comparar estas predicciones con los valores observados de esta variable en el conjunto de prueba;
4. Identificación de predicciones del conjunto de pruebas:
 - a. *FoldPredictions* que contiene las variables: número de *Fold*, *Line*, *Env*, *Observed* y *Predicted* para cada elemento del conjunto de prueba.
 - b. Cada fila de *FoldPrediction* se grega al data frame *Predictions*.

```

for (i in seq_along(folds)) {
  cat("\t*** Fold:", i, "****\n")
  fold <- folds[[i]]

  # Identify the training and testing sets
  X_training <- X[fold$training, ]
  X_testing <- X[fold$testing, ]
  y_training <- y[fold$training]
  y_testing <- y[fold$testing]

  # Model training
  model <- partial_least_squares(
    x = X_training,
    y = y_training
  )

  # Testing Predictions
  predictions <- predict(model, X_testing)

  # Predictions for the Fold Fold
  FoldPredictions <- data.frame(
    Fold = i,
    Line = PhenoToy$Line[fold$testing],
    Env = PhenoToy$Env[fold$testing],
    Observed = y_testing,
    Predicted = predictions$predicted
  )
  Predictions <- rbind(Predictions, FoldPredictions)
}

```

```

# Execution times
FoldTime <- data.frame(
  kernel = kernel,
  Fold = i,
  Minutes = as.numeric(model$execution_time, units = "mins")
)
Times <- rbind(Times, FoldTime)
}
#> *** Fold: 1 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.7359 secs ***
#> *** Fold: 2 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.5589 secs ***
#> *** Fold: 3 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.5334 secs ***
#> *** Fold: 4 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.5152 secs ***
#> *** Fold: 5 ***
#> *** Fitting Partial Least Squares model ***
#> *** Model evaluation completed in 0.6778 secs ***

```

Recordando que este proceso se realizó para cada combinación de tipo de kernel y relación de línea especificada, cada una de las listas *PredictionsAll*, *TimesAll* y *SummariesAll* contiene las predicciones, los tiempos de ejecución y los resúmenes, respectivamente, para *kernel_type* "Sparse_arc_cosie" y la relación de líneas aplicadas a la matriz de datos *X*.

```

head(Predictions)
#>   Fold      Line Env  Observed Predicted
#> 1     1 ICCV97301   6 109.66667 108.82027
#> 2     1 ICCV04103   1 106.66667 103.32402
#> 3     1 ICCV05109   4  95.00000  93.71077
#> 4     1 ICCV00402   7  93.33333 103.63993
#> 5     1 ICCV09114   4  96.00000  93.09728
#> 6     1 ICCV03102   2 152.50000 135.67279
unique(Predictions$Fold)
#> [1] 1 2 3 4 5

# Execution times
Times
#>      kernel Fold      Minutes
#> 1 arc_cosine   1 0.012265472
#> 2 arc_cosine   2 0.009314585
#> 3 arc_cosine   3 0.008889588
#> 4 arc_cosine   4 0.008586792
#> 5 arc_cosine   5 0.011297433
# Summaries

```

```

summaries <- gs_summaries(Predictions)

# Elements of summaries
names(summaries)
#> [1] "line" "env" "fold"
# Summaries by Line
head(summaries$line)
#>      Line Observed Predicted Difference
#> 1 ICCV08302 103.3889 103.4398 0.0510
#> 2 ICCV07310 107.3889 107.2986 0.0903
#> 3 ICCV03109 111.1250 111.2698 0.1448
#> 4 ICCV10112 113.0833 113.2894 0.2061
#> 5 ICCV07305 109.7000 109.2347 0.4653
#> 6 ICCV97301 122.7333 123.2577 0.5244

# Summaries by Environment
summaries$env[, 1:9]
#>      Env      MSE  MSE_SE  RMSE RMSE_SE  NRMSE NRMSE_SE  MAE
#> 1      1  2.3781  0.6565 1.4667  0.2381 0.8475  0.0517 1.1369
#> 2      2 28.0994 12.9687 4.5891  1.3266 2.1055  0.7556 3.4706
#> 3      4 27.7770  4.9394 5.1607  0.5349 1.9039  0.2641 4.0770
#> 4      5 22.0669  7.6682 4.1129  1.1347 1.9947  0.5884 3.5034
#> 5      6 24.6203  7.1163 4.7076  0.7840 1.1324  0.1475 4.0640
#> 6      7 17.3899  5.2171 3.8414  0.8114 2.2305  0.4400 2.7593
#> 7 Global 14.7980  3.0684 3.7569  0.4135 0.2009  0.0287 2.7441
#>      MAE_SE
#> 1 0.1899
#> 2 0.9939
#> 3 0.3820
#> 4 0.9240
#> 5 0.6905
#> 6 0.5081
#> 7 0.2851
summaries$env[, 10:17]
#>      Cor Cor_SE Intercept Intercept_SE  Slope Slope_SE
#> 1 0.6410 0.0947 -301.0291 284.5196 3.9480 2.7935
#> 2 0.5908 0.1263  54.9163  61.2350 0.6439 0.3927
#> 3 -0.0840 0.3112  87.6664  26.0151 0.0748 0.2788
#> 4 -0.3720 0.0815 143.3415  24.5397 -0.7051 0.2626
#> 5 -0.0499 0.2030 -724.3812 946.3272 7.7310 8.7799
#> 6 0.2317 0.1481  60.2321  28.8307 0.3502 0.3112
#> 7 0.9794 0.0049  -4.9145   2.1590 1.0449 0.0221
#>      R2  R2_SE
#> 1 0.4468 0.1403
#> 2 0.4128 0.1611
#> 3 0.3943 0.1757
#> 4 0.1633 0.0717
#> 5 0.1674 0.1013
#> 6 0.1415 0.0786
#> 7 0.9594 0.0096

```

```

summaries$env[, 18:19]
#>      MAAPE MAAPE_SE
#> 1 0.0110  0.0019
#> 2 0.0225  0.0064
#> 3 0.0435  0.0045
#> 4 0.0381  0.0102
#> 5 0.0369  0.0063
#> 6 0.0296  0.0054
#> 7 0.0263  0.0030

# Summaries by Fold
summaries$fold[, 1:9]
#>      Fold      MSE MSE_SE  RMSE RMSE_SE  NRMSE NRMSE_SE  MAE
#> 1      1 25.6326 10.4167 4.5062  1.0322 2.2430  0.5937 3.4801
#> 2      2 15.5967  5.2968 3.3264  0.9520 1.4066  0.2416 2.6256
#> 3      3 23.3940  7.8274 4.2212  1.0560 1.4994  0.2994 3.4327
#> 4      4 20.3442  6.6982 4.1100  0.8309 1.6025  0.4519 3.2363
#> 5      5 16.9756  6.3938 3.7349  0.7779 1.6626  0.4865 3.0679
#> 6 Global 14.7980  3.0684 3.7569  0.4135 0.2009  0.0287 2.7441
#>      MAE_SE
#> 1 0.7672
#> 2 0.7572
#> 3 0.8503
#> 4 0.7105
#> 5 0.6123
#> 6 0.2851
summaries$fold[, 10:17]
#>      Cor Cor_SE Intercept Intercept_SE  Slope Slope_SE
#> 1 -0.1006 0.2322  150.8418    49.9086 -0.4403  0.4800
#> 2  0.1938 0.1698   86.7755    28.1197  0.2287  0.2904
#> 3  0.3221 0.1998  -86.2070   288.8574  1.9516  2.7986
#> 4  0.0798 0.2318  -15.9287    82.4040  0.9541  0.7690
#> 5  0.3972 0.2230 -701.5265   750.8235  7.4970  6.9720
#> 6  0.9794 0.0049   -4.9145     2.1590  1.0449  0.0221
#>      R2 R2_SE
#> 1 0.2798 0.0800
#> 2 0.1760 0.0699
#> 3 0.3034 0.1479
#> 4 0.2751 0.1268
#> 5 0.4063 0.1676
#> 6 0.9594 0.0096
summaries$fold[, 18:19]
#>      MAAPE MAAPE_SE
#> 1 0.0320  0.0060
#> 2 0.0243  0.0074
#> 3 0.0326  0.0089
#> 4 0.0319  0.0072
#> 5 0.0305  0.0069
#> 6 0.0263  0.0030

```