



Universidad Complutense de Madrid
Master en Big Data y Business Analytics

MACHINE LEARNING CON R,
CLASIFICACION BINARIA

Alumna

Ivonne V. Yáñez Mendoza

Profesor

Javier Portela

Enero de 2023

Índice

1	Introducción	1
2	Análisis exploratorio de los datos	2
3	Transformaciones	4
4	Selección de variables	4
5	Algoritmos de ML para clasificación binaria	7
5.1	Redes neuronales	7
5.1.1	Tunning de redes neuronales	7
5.1.2	Tunning de redes neuronales con base accuracy	7
5.1.3	Tunning de redes neuronales con MAXIT	8
5.1.4	Validación cruzada repetida para redes neuronales	9
5.1.5	Comparación de medias y boxplot	9
5.2	Bagging	10
5.2.1	Tunning de bagging	10
5.2.2	Validacion cruzada repetida para bagging	11
5.3	Random Forest Classifier	13
5.3.1	Tunning de random forest	13
5.3.2	Validacion cruzada para random forest	15
5.3.3	Comparacion de medias y boxplot	15
5.4	Stochastic Gradient Boosting	16
5.4.1	Tunning de gradient boosting	16
5.4.2	Validacion cruzada para gradient boosting	18
5.4.3	Comparacion de medias y boxplot	18
5.5	Xgboost	19
5.5.1	Tunning de Xgboost	19
5.5.2	Comparacion de medias y boxplot	21
5.6	Support Vector Machines	22
5.6.1	Tunning de support vector machines o SVM	22
5.6.2	Validacion cruzada para support vector machines	25
5.6.3	Comparacion de medias y boxplot	26
6	Ensamblado de algoritmos	26
6.1	Preparacion del ensamblado con <i>caret ensemble</i>	26
6.2	Validacion cruzada repetida y boxplot	28
6.2.1	Paso 1, carga del archivo con la funcion “cruzadas ensamblado binaria fuente.R”	29
6.2.2	Paso 2, Preparacion del archivo	29
6.2.3	Paso 3, aplicacion de la funcion cruzadas ensamblado	29
6.2.4	Paso 4, construccion del ensamblado	29
6.2.5	Paso 5, Procesado de los ensamblados	29
6.2.6	Paso 6, boxplot inicial	30
6.2.7	Paso 7, tabla con resultados	30
6.2.8	Paso 8, Boxplot ordenados para comparacion de medias	31
6.2.9	Paso 9, comparacion de mejores modelos	32
6.2.10	Paso 10, revision a los mejores ensamblados	33
6.2.11	Observaciones finales	33

7	Evaluacion del modelo ganador y conclusiones	33
7.1	Algoritmo a usar para este modelo	33
7.2	Matriz de confusion para el modelo escogido	34
7.3	Sensitividad, especificidad y precision	34
7.4	Tabla de parametros para regresion logistica	34
7.5	Puntos de corte	34
7.6	Contraste de hipotesis entre algunos modelos	35
7.7	Visualpred	35
A	Anexos	i
A.1	Otros metodos de seleccion de variables	i
A.1.1	Utilizando la libreria party y random forest	i
A.1.2	Utilizando <i>MARS: Multivariate Adaptive Regression Splines</i>	i
A.1.3	Utilizando la libreria Boruta	ii
A.1.4	Utilizando RFE con caret	ii
A.2	Arboles de clasificacion	iii
A.2.1	Seleccion de variables	iii
A.2.2	Tunning de algoritmos usando rpart()	v
A.2.3	Tunning de algoritmos usando caret()	vi
A.2.4	Comparacion con otros algoritmos de ML	viii
A.3	Libreria h2o para python	ix

1 Introducción

Para este trabajo de clasificación binaria utilizando diversas técnicas de machine learning, se ha decidido utilizar el set de datos *Dengue prevalence by administrative region*.

Este dataset contiene información relevante sobre la prevalencia de la enfermedad por cada región administrativa de Nueva Zelanda (2000 regiones), es decir, si se ha observado la presencia del dengue o no, desde 1961 hasta 1990.

El objetivo de este trabajo es entrenar diversos algoritmos de clasificación, con la finalidad de predecir una variable binaria y de tomar una decisión sobre cual modelo propuesto es el mas recomendado para el tipo de dataset con el que se está trabajando.

Fuentes

1. Repositorio: <https://vincentarelbundock.github.io/Rdatasets/datasets.html>
2. Explicación de los datos (en ingles):
<https://vincentarelbundock.github.io/Rdatasets/doc/DAAG/dengue.html>
3. Descripción de las columnas que contiene el dataset dengue:
 - 3.1 **humid**: Densidad de vapor promedio desde 1961 a 1990.
 - 3.2 **humid90**: Percentil 90 para humedad.
 - 3.3 **temp**: Temperatura promedio desde 1961 a 1990.
 - 3.4 **temp90**: Percentil 90 para temperatura.
 - 3.5 **h10pix**: Humedad máxima dentro de un radio de 10 pixeles.
 - 3.6 **h10pix90**: Humedad máxima de variable temp90 dentro de un radio de 10 pixeles.
 - 3.7 **trees**: Porcentaje de un área cubierta por arboles, dato entregado por satélite.
 - 3.8 **trees90**: Percentil 90 de la variable trees.
 - 3.9 **NoYes**: Se ha observado la presencia de dengue, 1 indica si. **Variable dependiente a estudiar.**
 - 3.10 **Xmin**: Longitud mínima.
 - 3.11 **Xmax**: Longitud máxima.
 - 3.12 **Ymin**: Latitud mínima.
 - 3.13 **Ymax**: Latitud máxima.
 - 3.14 **X**: Indicador fila.
4. Dimensión de los datos: 2000 filas x 14 columnas

2 Analisis exploratorio de los datos

X	humid	humid90	temp	temp90	h10pix	h10pix90	trees	trees90	NoYes	Xmin	Xmax	Ymin	Ymax
1	0.6713889	4.416667	2.037500	8.470835	17.35653	17.80861	0	1.5	0	70.5	74.5	38.0	35.5
2	7.6483340	8.167499	12.325000	14.925000	10.98361	11.69167	0	1.0	0	62.5	64.5	35.5	34.5
3	6.9790556	9.563057	6.925000	14.591660	17.50833	17.62528	0	1.2	0	68.5	69.5	36.0	35.0
4	1.1104163	1.825361	4.641665	6.046669	17.41764	17.51694	0	0.6	0	67.0	68.0	35.0	34.0
5	9.0270555	9.742751	18.175000	19.710000	13.84306	13.84306	0	0.0	0	61.0	64.5	33.5	32.0
6	8.9141113	9.516778	11.900000	16.643341	11.69167	11.69167	0	0.2	0	64.5	65.5	36.5	35.0

```
'data.frame': 2000 obs. of 14 variables:
 $ X      : int  1 2 3 4 5 6 7 8 9 10 ...
 $ humid  : num  0.671 7.648 6.979 1.11 9.027 ...
 $ humid90: num  4.42 8.17 9.56 1.83 9.74 ...
 $ temp   : num  2.04 12.32 6.93 4.64 18.18 ...
 $ temp90 : num  8.47 14.93 14.59 6.05 19.71 ...
 $ h10pix : num  17.4 11 17.5 17.4 13.8 ...
 $ h10pix90: num  17.8 11.7 17.6 17.5 13.8 ...
 $ trees  : num  0 0 0 0 0 0 0 0 0 0 ...
 $ trees90: num  1.5 1 1.2 0.6 0 ...
 $ NoYes  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ Xmin   : num  70.5 62.5 68.5 67 61 64.5 67.5 64 63.5 61 ...
 $ Xmax   : num  74.5 64.5 69.5 68 64.5 65.5 68.5 66.5 65.5 64.5 ...
 $ Ymin   : num  38 35.5 36 35 33.5 36.5 33.5 35 33 35 ...
 $ Ymax   : num  35.5 34.5 35 34 32 35 32 33 29.5 33.5 ...
```

[1] 2000 14

Como se observa anteriormente, son 14 columnas con 2000 filas, los tipos de datos son de tipo numérico incluyendo la variable objetivo **NoYes**, se realiza a continuación una exploración mas detallada del dataset a estudiar.

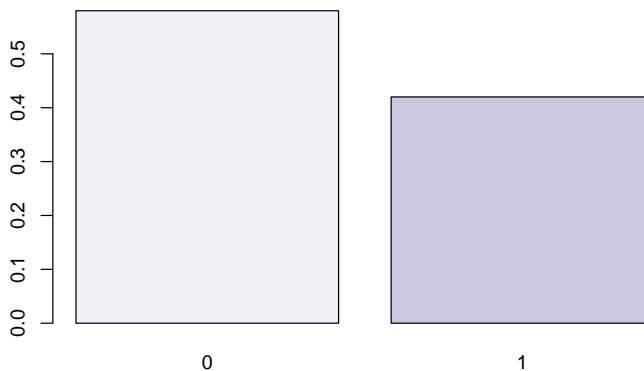
En cuanto a las observaciones sobre la variable independiente (*NoYes*) y observando el gráfico y tabla referencia, se tiene:

Para un total de 831 observaciones en la clase minoritaria con 12 variables input (se omite columna X y la objetivo) 69 observaciones por parámetro. Respecto a los porcentajes y sobretodo el porcentaje de observaciones de la clase minoritaria da una idea de las métricas a tener en cuenta en la evaluación posterior de los modelos, es decir, un accuracy mayor a 0.58 y una tasa de fallos menor a 0.41 debería mejorar el resultado base.

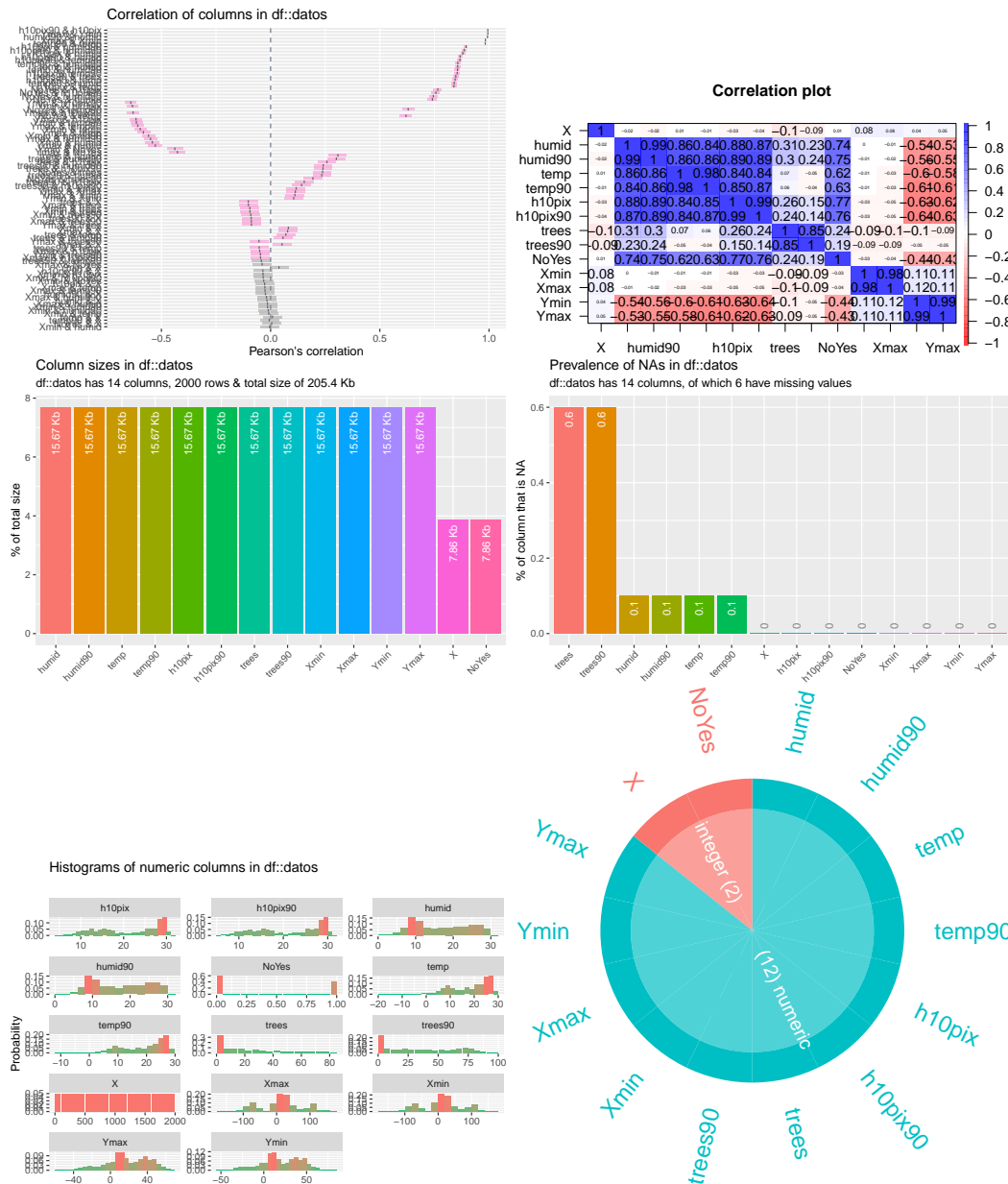
Se observa un ligero desbalance el cual no debería afectar el proceso de modelado, no se aplican técnicas de balanceo artificial de los datos.

NoYes	cuenta	porcentaje
0	1169	0.58
1	831	0.42

Distribucion en % variable dependiente NoYes



A continuación se presentan algunos gráficos de apoyo al análisis exploratorio:



Observaciones:

- Correlación:
- Tipos de datos por columna: Datos tipo numeric e integer incluyendo la variable estudio.
- NA's: No se presenta un % alto de datos ausentes, pero deben ser tratados para evitar problemas en el proceso, en la siguiente sección se trabajará en ello. trees y trees90 presentan un NA cercano al 0.6%
- Histograma: Se observan variables que no siguen una distribución normal, X y NoYes deben ser separadas del dataset (X por no ser relevante es un índice y NoYes es la objetivo)

3 Transformaciones

Se realizan transformaciones sobre los datos, eliminación de columnas con escasa relevancia, tratamiento de datos faltantes y estandarización de los datos continuos:

```
# Se elimina la columna X
datos$X <- NULL

# Tratamiento de datos faltantes, se eliminan
datos2 <- na.omit(datos, (is.na(datos)))
colSums(is.na(datos2)) > 0

      humid humid90    temp    temp90   h10pix h10pix90   trees trees90
FALSE     FALSE    FALSE    FALSE    FALSE    FALSE    FALSE    FALSE
NoYes      Xmin     Xmax     Ymin     Ymax
FALSE     FALSE    FALSE    FALSE    FALSE

# Se separa la variable objetivo de la variable input
varObjBin <- datos2$NoYes
input <- as.data.frame(datos2[, -(9)])

# Pre procesado, recategorizar, dummies, estandarizar etc
# Estandarizacion
temp <- input %>% mutate_if(is.numeric, scale)
head(temp)

      humid humid90    temp    temp90   h10pix h10pix90   trees
1 -2.187817 -1.752592 -2.04603997 -1.42684042 -0.5252414 -0.5182986 -0.9516725
2 -1.235570 -1.240199 -0.76267048 -0.58677814 -1.3943124 -1.3621731 -0.9516725
3 -1.326917 -1.049555 -1.43632247 -0.63016510 -0.5045405 -0.5435904 -0.9516725
4 -2.127896 -2.106585 -1.72116940 -1.74236542 -0.5169085 -0.5585368 -0.9516725
5 -1.047396 -1.025007 -0.03288072  0.03602869 -1.0043711 -1.0653740 -0.9516725
6 -1.062811 -1.055877 -0.81568941 -0.36312211 -1.2977547 -1.3621731 -0.9516725
      trees90      Xmin      Xmax      Ymin      Ymax
1 -1.173965  0.9177821  0.9477775  0.7759536  0.7328488
2 -1.191357  0.7886923  0.7859356  0.6691044  0.6904187
3 -1.184400  0.8855096  0.8668566  0.6904742  0.7116337
4 -1.205271  0.8613053  0.8425803  0.6477345  0.6692036
5 -1.226141  0.7644880  0.7859356  0.5836250  0.5843434
6 -1.219184  0.8209648  0.8021198  0.7118441  0.7116337

# Se une la variable objetivo con los resultados(de estandarizar, dummies etc)
base <- data.frame(varObjBin, temp)

# La variable objetivo se cambia a alfanumerico yes, no
base$varObjBin <- ifelse(base$varObjBin == 1, "Yes", "No")
```

Los datos han sido preparados y transformados, se continua con la selección de variables

4 Selección de variables

Se realiza una selección automática de variables de tipo stepwise para escoger las features que son realmente relevantes para el posterior proceso de modelamiento y *tunning* de algoritmos.

```
# Selección de variables ----
full <- glm(factor(varObjBin) ~., data = base, family = binomial(link = "logit"))
null <- glm(factor(varObjBin) ~ 1, data = base, family = binomial(link = "logit"))

# Selección de variables automatica
seleccion <- stepAIC(null, scope = list(upper = full), direction = "both", trace = F)

# Para ver las features escogidas de forma automatica
dput(names(seleccion$coefficients))

c("(Intercept)", "h10pix", "temp90", "Ymin", "Ymax", "temp",
  "humid", "humid90", "trees", "trees90")

# Version formula
formula(seleccion)

factor(varObjBin) ~ h10pix + temp90 + Ymin + Ymax + temp + humid +
  humid90 + trees + trees90
summary(seleccion)

Call:
glm(formula = factor(varObjBin) ~ h10pix + temp90 + Ymin + Ymax +
  temp + humid + humid90 + trees + trees90, family = binomial(link = "logit"),
  data = base)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.63747  -0.15358  -0.02882   0.44958   3.04419

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept)  -1.8848      0.1866 -10.102 < 0.0000000000000002 ***
```

```

h10pix      3.1524      0.2542     12.403 < 0.0000000000000002 ***
temp90      3.1225      0.7286      4.286      0.0000182 ***
Ymin        -1.8854      1.1527     -1.636      0.101914
Ymax        1.6698      1.1480      1.455      0.145781
temp        -2.6276      0.6948     -3.782      0.000156 ***
humid       3.7731      0.9470      3.984      0.0000677 ***
humid90     -3.4328      0.9786     -3.508      0.000452 ***
trees       -0.6970      0.1991     -3.500      0.000465 ***
trees90      0.5499      0.2221      2.477      0.013266 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 2695.4  on 1985  degrees of freedom
Residual deviance: 1033.8  on 1976  degrees of freedom
AIC: 1053.8

Number of Fisher Scoring iterations: 7

```

Según el código anterior, las variables relevantes seleccionadas son: h10pix + temp90 + Ymin + Ymax + temp + humid + humid90 + trees + trees90, con las cuales se aplica la función `steprepetidobinaria()` con criterio “AIC” para remuestreo:

```

# Selección AIC
listaAIC <- steprepetidobinaria(
  data = data,
  vardep = vardep,
  listconti = listconti,
  inicio = 12345,
  sfinal = 12355,
  porcen = 0.8,
  criterio = "AIC")

tabla1 <- listaAIC[[1]]
knitr::kable(tabla1, "pipe")

```

	modelo	Freq	contador
1	h10pix+temp90+trees+trees90+Ymin+Ymax+temp+humid+humid90	4	9
5	h10pix+temp90+trees+trees90+Ymin+temp+humid+humid90	3	8
8	h10pix+temp90+trees+trees90+temp+humid+humid90	2	7
10	h10pix+temp90+trees+trees90+Ymin+Ymax	1	6
11	h10pix+trees+Ymin+humid	1	4

```
dput(listaAIC[[2]][[1]])
```

```

c("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax", "temp",
  "humid", "humid90")
dput(listaAIC[[2]][[2]])

```

```

c("h10pix", "temp90", "Ymin", "temp", "humid", "humid90", "trees",
  "trees90")

```

Se repite el mismo proceso pero esta vez con criterio “BIC”:

```

# Selección con Bic
listaBIC <- steprepetidobinaria(
  data = data,
  vardep = vardep,
  listconti = listconti,
  inicio = 12345,
  sfinal=12355, porcen = 0.8, criterio = "BIC")

tabla2 <- listaBIC[[1]]
knitr::kable(tabla2, "pipe")

```

	modelo	Freq	contador
1	h10pix+temp90	8	2
9	h10pix+humid	1	2
10	h10pix+humid+trees	1	3
11	h10pix+temp90+Ymin	1	3

```
dput(listaBIC[[2]][[1]])
```

```

c("h10pix", "temp90")
dput(listaBIC[[2]][[2]])

```

```
c("h10pix", "humid")
```

Observando los resultados anteriores y obteniendo los set de variables se comparan con validación cruzada repetida:


```

# Selección de variables AIC
medias1 <- cruzadalogistica(
  data = data,
  vardep = "varObjBin",
  listconti = c("h10pix", "temp90", "trees", "trees90",
    "Ymin", "Ymax", "temp", "humid", "humid90"),
  listclass = c(""),
  grupos = 4,
  inicio = 1234,
  repe = 5)

medias1$modelo <- "Logística1"

# Selección de variables BIC
medias2 <- cruzadalogistica(
  data = data,
  vardep="varObjBin",
  listconti = c("h10pix", "temp90"),
  listclass = c(""),
  grupos = 4,
  inicio = 1234,
  repe = 5)

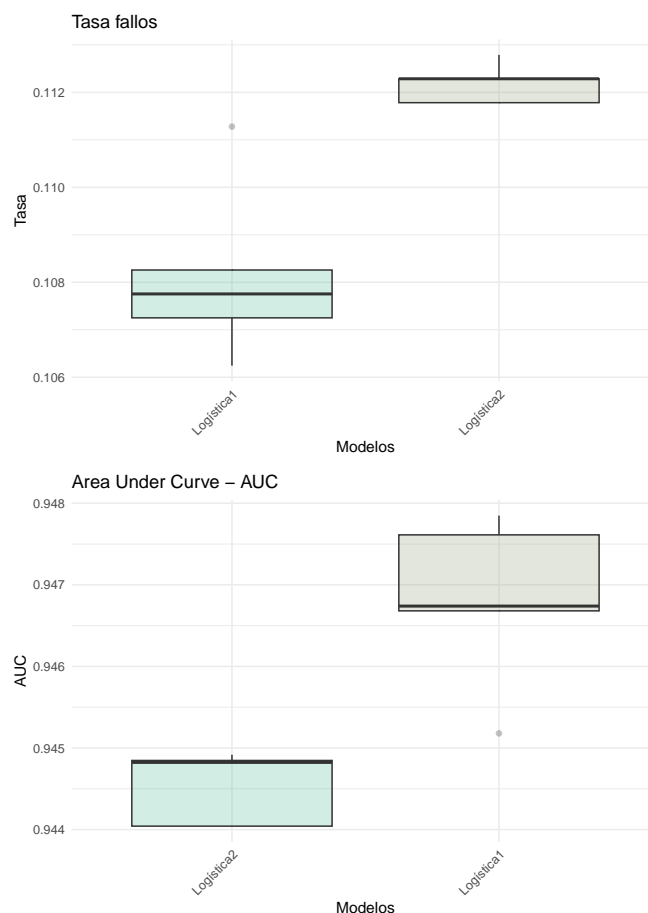
medias2$modelo <- "Logística2"

```

La función `genera_graficos()` toma las medias de los modelos generados y entrega el respectivo gráfico de tipo boxplot para facilitar la visualización: Se puede revisar en el siguiente repositorio:

https://github.com/TiaIvonne/Machine-LearningR/blob/master/src/functions/genera_graficos.R

```
genera_graficos(medias1, medias2)
```



Observaciones:

- El modelo de Logística1 con las variables elegidas con criterio AIC entrega mejores resultados que el modelo generado con BIC en AUC y Tasa de fallos respectivamente. De acuerdo a lo anterior, las variables elegidas con criterio AIC son las que se trabajarán en los modelos siguientes.

5 Algoritmos de ML para clasificación binaria

5.1 Redes neuronales

5.1.1 Tuning de redes neuronales

Una vez seleccionada las variables utilizando stepwise visto en la sección anterior, se comienza con el tuning de redes neuronales para un problema de clasificación binaria con la finalidad de encontrar los parámetros que se ajusten mejor para el dataset con el que se está trabajando, realizar un nuevo modelamiento y comparar en este caso con regresión logística y mas adelante, con los próximos algoritmos a modelar (Bagging, Random Forest etc).

5.1.2 Tuning de redes neuronales con base accuracy

Utilizando las variables seleccionadas bajo StepwiseAIC se prueba un primer modelo:

```
set.seed(12345)
control_redes <- trainControl(method = "repeatedcv", number = 4, repeats = 5,
  savePredictions = "all", classProbs = TRUE)

avnnnetgrid <- expand.grid(
  size = c(5, 10, 15, 20),
  decay = c(0.01, 0.1, 0.001), bag = FALSE)

# Con las variables seleccionadas stepwiseAIC
redavnnnet1 <- train(
  varObjBin ~ h10pix + temp90 + trees + trees90 + Ymin + Ymax + temp +
  humid + humid90,
  data = dengue,
  method = "avNNet",
  linout = FALSE,
  maxit = 100,
  trControl = control_redes,
  tuneGrid = avnnnetgrid,
  repeats = 5,
  trace = F)

knitr::kable(redavnnnet1$results, "pipe")
```

Con el código anterior se ha generado la siguiente tabla la cual puede ser de utilidad para determinar cuales parámetros podrían ser útiles al momento de elegir para entrenar el modelo:

size	decay	bag	Accuracy	Kappa	AccuracySD	KappaSD
5	0.001	FALSE	0.9052328	0.8075327	0.0139857	0.0278779
5	0.010	FALSE	0.9229599	0.8423037	0.0131987	0.0269257
5	0.100	FALSE	0.9179245	0.8325153	0.0104558	0.0211637
10	0.001	FALSE	0.9207424	0.8384348	0.0132498	0.0265557
10	0.010	FALSE	0.9260815	0.8484666	0.0098502	0.0203012
10	0.100	FALSE	0.9197360	0.8361295	0.0099104	0.0201965
15	0.001	FALSE	0.9211444	0.8389624	0.0114062	0.0230180
15	0.010	FALSE	0.9266882	0.8494893	0.0102616	0.0213015
15	0.100	FALSE	0.9225561	0.8418606	0.0098902	0.0201859
20	0.001	FALSE	0.9244712	0.8454846	0.0101698	0.0207285
20	0.010	FALSE	0.9274934	0.8511611	0.0100521	0.0207691
20	0.100	FALSE	0.9216511	0.8400228	0.0101441	0.0207448

Observando la tabla y con base el Accuracy, los parámetros a considerar son un size de 10 con un decay de 0.010.

Se prueba el modelo una vez mas utilizando las variables seleccionadas con criterio BIC (son solo dos las predictoras mas la variable objetivo)

```
# Con las variables seleccionadas BIC
redavnnnet2 <- train(
  varObjBin ~ h10pix + temp90 ,
  data = data,
  method = "avNNet", linout = FALSE, maxit = 100,
  trControl = control_redes, tuneGrid = avnnnetgrid,
  repeats = 5,
  trace = F)

knitr::kable(redavnnnet2$results, "pipe")
```

La siguiente tabla muestra los resultados con las variables obtenidas con BIC

size	decay	bag	Accuracy	Kappa	AccuracySD	KappaSD
5	0.001	FALSE	0.9052328	0.8075327	0.0139857	0.0278779
5	0.010	FALSE	0.9229599	0.8423037	0.0131987	0.0269257
5	0.100	FALSE	0.9179245	0.8325153	0.0104558	0.0211637
10	0.001	FALSE	0.9207424	0.8384348	0.0132498	0.0265557
10	0.010	FALSE	0.9260815	0.8484666	0.0098502	0.0203012
10	0.100	FALSE	0.9197360	0.8361295	0.0099104	0.0201965
15	0.001	FALSE	0.9211444	0.8389624	0.0114062	0.0230180
15	0.010	FALSE	0.9266882	0.8494893	0.0102616	0.0213015
15	0.100	FALSE	0.9225561	0.8418606	0.0098902	0.0201859
20	0.001	FALSE	0.9244712	0.8454846	0.0101698	0.0207285
20	0.010	FALSE	0.9274934	0.8511611	0.0100521	0.0207691
20	0.100	FALSE	0.9216511	0.8400228	0.0101441	0.0207448

Agregar observaciones

5.1.3 Tuning de redes neuronales con MAXIT

Otro criterio para realizar tuning de redes neuronales es utilizando un bucle for() que recorre un vector llamado listaiter que contiene las iteraciones a evaluar, buscando nuevamente los parámetros que podrían ser mas adecuados para este set de datos

```
listconti = c("h1Opix", "temp90", "trees", "trees90", "Ymin", "Ymax",
              "temp", "humid", "humid90")

data2 <- dengue[,c(listconti, vardep)]

control_maxit <- trainControl(method = "repeatedcv", number = 4, repeats = 5,
                              savePredictions = "all", classProbs = TRUE)

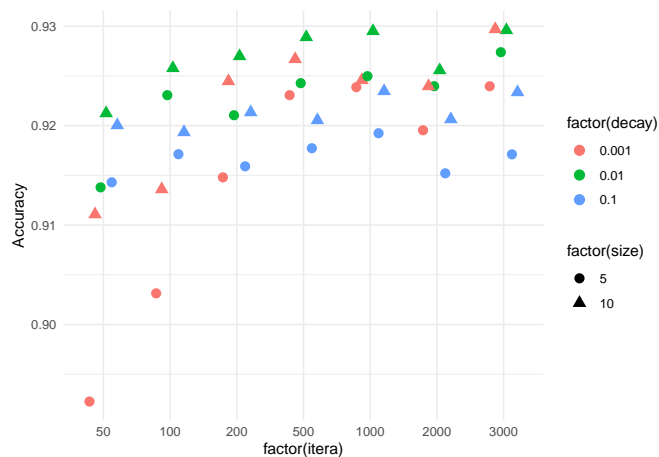
set.seed(12345)
nnetgrid <- expand.grid(size = c(5, 10), decay = c(0.01, 0.1, 0.001), bag = F)

completo <- data.frame()
listaiter <- c(50, 100, 200, 500, 1000, 2000, 3000)

for( iter in listaiter)
{
  rednnet <- train(
    varObjBin ~ .,
    data = data2,
    method = "avNNet",
    linout = FALSE,
    maxit = iter,
    trControl = control_redes,
    repeats = 5,
    tuneGrid = nnetgrid,
    trace = F)
  # Añado la columna del parametro de iteraciones
  rednnet$results$itera <- iter
  # Voy incorporando los resultados a completo
  completo <- rbind(completo, rednnet$results)
}

completo <- completo[order(completo$Accuracy),]

ggplot(completo, aes(x = factor(itera), y = Accuracy,
                    color = factor(decay), pch = factor(size))) +
  theme_minimal() +
  geom_point(position = position_dodge(width = 0.5), size = 3)
```



Observando el gráfico con los resultados del tuning ajustando maxit, los mejores resultados en

accuracy los entrega con un size de 10. En cuanto al decay muestra mejor comportamiento con decay 0.001 y 0.1. Con las iteraciones los mejores resultados los muestra entre 500 y 3000 aunque por simpleza con 500 iteraciones esta bien.

5.1.4 Validación cruzada repetida para redes neuronales

A continuación y una vez obtenidos los parámetros mas adecuados utilizando tuning con criterio accuracy y controlando maxit = se generan las correspondientes validaciones cruzadas repetidas utilizando la función `cruzadaavnnnetbin()`

```
# Validacion cruzada repetida con criterio accuracy
medias3 <-cruzadaavnnnetbin(
  data = dengue,
  vardep = "varObjBin",
  listconti = c("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax",
    "temp", "humid", "humid90"),
  listclass = c(""),
  grupos = 4,
  inicio = 1234,
  repe = 5,
  size = c(10),
  decay = c(0.010),
  repeticiones = 5,
  itera = 200)

medias3$modelo <- "Avnnnet1"

medias4 <- cruzadaavnnnetbin(
  data = dengue,
  vardep = "varObjBin",
  listconti = c("h10pix", "temp90"),
  listclass = c(""),
  grupos = 4,
  inicio = 1234,
  repe = 5,
  size = c(15),
  decay = c(0.10),
  repeticiones = 5,
  itera = 200)

medias4$modelo <- "Avnnnet2"

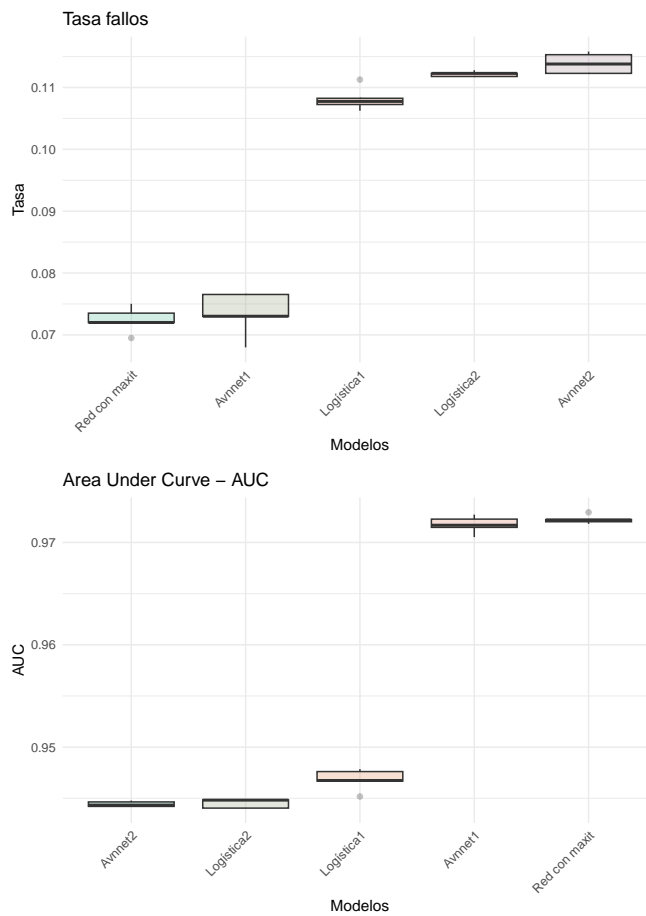
# Validacion cruzada repetida tuning maxit
medias5 <- cruzadaavnnnetbin(
  data = data2,
  vardep = "varObjBin",
  listconti = c("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax",
    "temp", "humid", "humid90"),
  listclass = c(""),
  grupos = 4,
  inicio = 1234,
  repe = 5,
  repeticiones = 5,
  itera = 1000,
  size = c(10),
  decay = c(0.01))

medias5$modelo <- "Red con maxit"
```

5.1.5 Comparación de medias y boxplot

Utilizando la función `genera_gráficos()` se generan los gráficos de tipo boxplot para comparar los modelos creados.

```
genera_graficos(medias1, medias2, medias3, medias4, medias5)
```



Observaciones:

- Si se compara con los modelos generados bajo regresión logística, hay un aumento considerable del criterio AUC con redes neuronales, específicamente la que se obtuvo con las variables seleccionadas con criterio AIC mas el modelo que utilizo los parámetros que entrego el tuning con maxit.
- Recalcar que es una comparativa inicial y aún se deben entrenar nuevos algoritmos y modelados para tomar una decisión.

5.2 Bagging

5.2.1 Tuning de bagging

En este apartado se probaran algunas técnicas para realizar tuning de bagging, partiendo de un modelo “básico” utilizando las variables previamente seleccionadas con stepwise y al final de este apartado realizando pruebas de tuning sobre el tamaño muestral del dataset dengue.

Primer modelo de bagging con variables previamente seleccionadas:

```
rfgrid <- expand.grid(mtry = c(6))
set.seed(12345)
control_bagging <- trainControl(method = "repeatedcv", number = 4, repeats = 5,
  savePredictions = "all", classProbs = TRUE)

bagging <- train(
  data = dengue,
  factor(varObjBin) ~ h10pix + temp90 + trees + trees90 + Ymin + Ymax + temp +
  humid + humid90,
  method = "rf",
  trControl = control_bagging,
  tuneGrid = rfgrid,
  linout = FALSE,
  ntree=5000,
```

```
sampsize = 200,
nodesize = 10,
replace = TRUE)
```

```
bagging$modelType
```

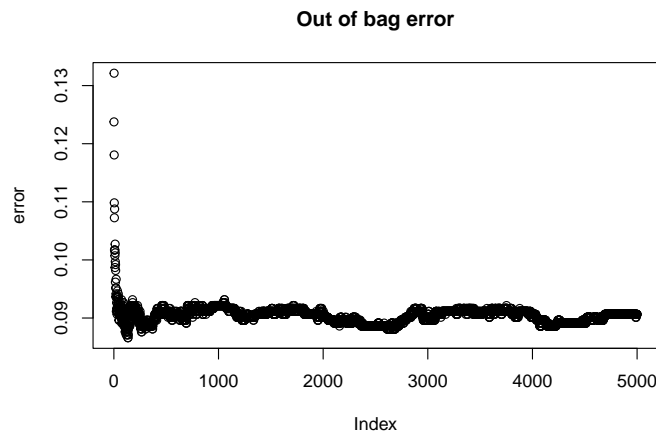
mtry	Accuracy	Kappa	AccuracySD	KappaSD
6	0.8987865	0.7945759	0.0143582	0.0286637

A continuación se explora el Out of bag error para este modelo, buscando parámetros para optimización, tal como lo indican los apuntes, se debe utilizar la función de `randomForest()` para extraer los datos y presentar gráfico:

```
# Plotear
baggingbis <- randomForest(
  factor(varObjBin) ~ h10pix + temp90 + trees + trees90 + Ymin + Ymax + temp +
  humid + humid90,
  data = dengue,
  mtry = 6,
  ntree = 5000,
  sampsize = 300,
  nodesize = 10,
  replace = TRUE)
```

```
#Se plotea la columna 1 que es OOB
plot(baggingbis$err.rate[, 1], main = "Out of bag error", ylab = "error")
```

```
# 1.3 Validacion cruzada rep bagging -----
```



A partir de las 1000 muestras se estabiliza el error de los datos analizados

5.2.2 Validacion cruzada repetida para bagging

Se realizan algunas pruebas con validación cruzada repetida utilizando la función `cruzadarfbin()` y cuidando de incluir el parámetro `mtry`

```
# Con validacion cruzada y 4 grupos
medias6 <- cruzadarfbin(
  data = dengue,
  vardep = "varObjBin",
  listconti = c("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax",
  "temp", "humid", "humid90"),
  listclass = c(""),
  grupos = 4,
  sinicio = 1234,
  repe = 5,
  mtry = 6,
  ntree = 1000,
  replace = TRUE)

medias6$modelo <- "bagging"
```

A continuación y a modo de complemento se prueba a manipular el tamaño muestral para observar el efecto sobre bagging utilizando el parámetro `sampsize`. Para el dataset *dengue* son 2000 observaciones con 4 grupos de validación cruzada cada grupo contara con 1500 observaciones para probar:

Nota: El bagging medias7 o bagging_base es solo como adicional pues se hace con 10 grupos de validación cruzada, es solo para observar el efecto y comparar. medias8, medias9 y medias10 si se consideran pues utilizan 4 grupos de validación y con diversos tamaños muestrales:

```
# Anexo Manipulando tamaño muestral con 10 grupos no incluir en la selección
# de modelos pues el grupos = es diferente
medias7 <-cruzadarfbn(
  data = dengue,
  vardep = "varObjBin",
  listconti = c("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax",
    "temp", "humid", "humid90"),
  listclass = c(""),
  grupos = 10,
  inicio = 1234,
  repe=20,
  nodesize = 10,
  mtry = 6,
  ntree = 3000,
  replace = TRUE)

medias7$modelo <- "bagging_base"

# Bagging manipulando sampsize 1000
medias8 <-cruzadarfbn(
  data = dengue,
  vardep = "varObjBin",
  listconti = c("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax",
    "temp", "humid", "humid90"),
  listclass = c(""),
  grupos = 10,
  inicio = 1234,
  repe=20,
  nodesize = 10,
  mtry = 6,
  ntree = 3000,
  replace = TRUE,
  sampsize = 1000 )

medias8$modelo <- "bagging1000"

# Bagging manipulando sampsize 1250
medias9 <-cruzadarfbn(
  data = dengue,
  vardep = "varObjBin",
  listconti = c("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax",
    "temp", "humid", "humid90"),
  listclass = c(""),
  grupos = 10,
  inicio = 1234,
  repe=20,
  nodesize = 10,
  mtry = 6,
  ntree = 3000,
  replace = TRUE,
  sampsize = 1250 )

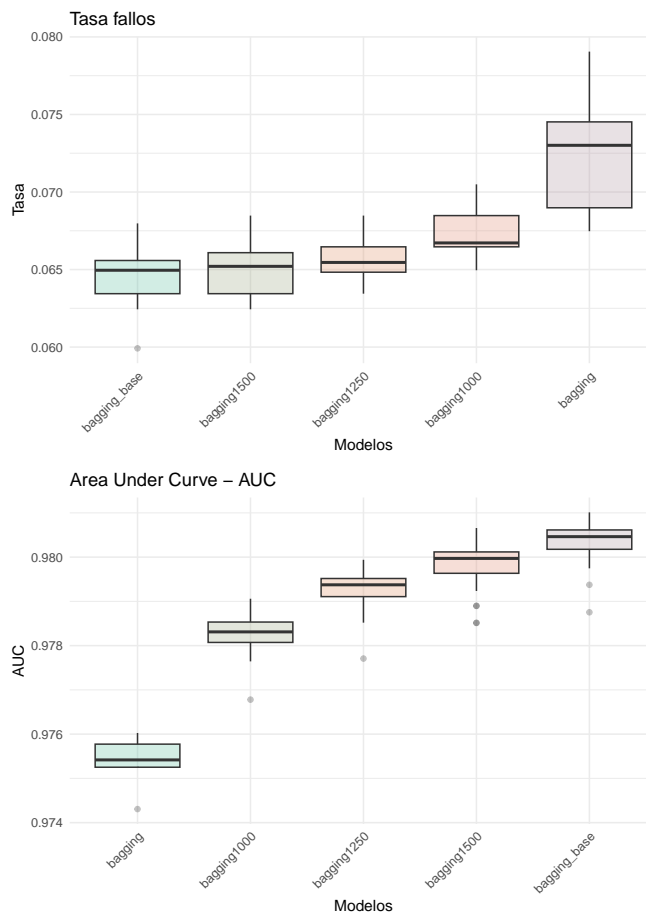
medias9$modelo <- "bagging1250"

# Bagging manipulando sampsize 1500
medias10 <-cruzadarfbn(
  data = dengue,
  vardep = "varObjBin",
  listconti = c("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax",
    "temp", "humid", "humid90"),
  listclass = c(""),
  grupos = 10,
  inicio = 1234,
  repe=20,
  nodesize = 10,
  mtry = 6,
  ntree = 3000,
  replace = TRUE,
  sampsize = 1500 )

medias10$modelo <- "bagging1500"
```

Como parte de este complemento se evalúan las medias con la función genera_gráficos() para comparar cual de todos estos tamaños muestrales entrega mejores métricas:

```
# genera graficos para comparar tamaño muestral de bagging
genera_graficos(medias6, medias7, medias8, medias9, medias10, medias10)
```



Observaciones

- Observando todo el grupo evaluado y descartando `bagging_base` que no se considera, los mejores resultados están entre `bagging 1250` y `bagging 1500`. Para comparar modelos en el apartado final incluyendo los modelos de random forest se deja el modelo `medias9`.

5.3 Random Forest Classifier

5.3.1 Tuning de random forest

Primer modelo utilizando todas las variables y buscando las variables predictoras mas relevantes.

Nota: Solo como complemento a la practica, para observar como se comporta el algoritmo al seleccionar variables. Para comparar medias, validación cruzada y probar con diferentes números de arboles se utilizan las variables seleccionadas con *stepwise*

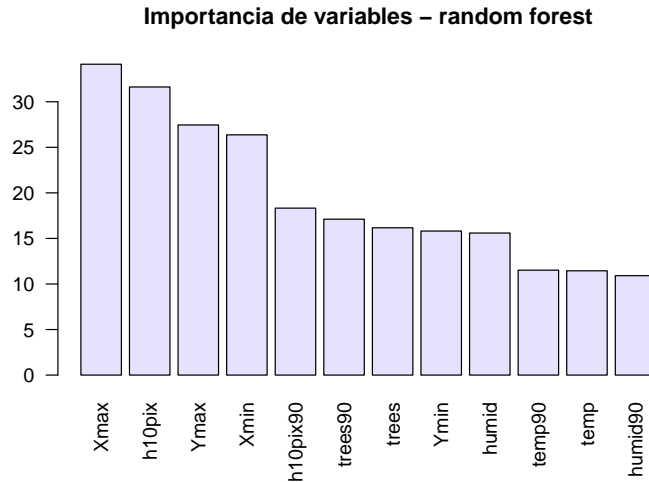
```
# 2. Tuneo random forest -----
set.seed(12345)
rfgrid_rf <- expand.grid(mtry = c(3, 4, 5, 6, 7, 8, 9, 10, 11))

control_rf <- trainControl(method = "repeatedcv", number = 4, repeats = 5,
  savePredictions = "all", classProbs = TRUE)

random_forest <- train(
  factor(varObjBin) ~.,
  data = dengue,
  method = "rf",
  trControl = control_rf,
  tuneGrid = rfgrid_rf,
  linout = FALSE,
  ntree = 300,
  nodesize = 10,
  replace = TRUE,
  importance = TRUE)

# Mejor valor de mtry
random_forest$bestTune$mtry
```


Se obtienen los resultados de este primer modelo de randomforest



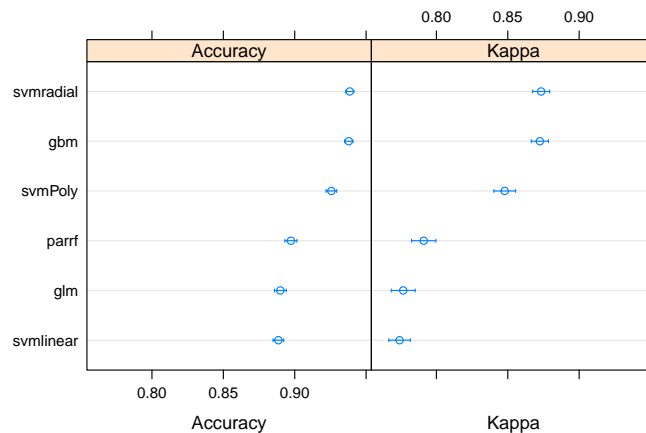
Para el modelo anterior el mejor mtry es 9 y en el gráfico se observa que las variables predictoras mas relevantes serian: Xmax, h10pix, Ymax, trees90, h10pix90, humid, Ymin lo cual difieren a los resultados entregados por la selección *stepwise* pero para explorar las alternativas que ofrece el algoritmo es útil.

A continuación se prueba generando un bucle for() para buscar los mejores indicadores evaluando según el mtry y numero de trees, desde 300 a 2500 para observar su comportamiento y buscar los mejores parámetros para la validación cruzada:

```
# random forest solo usando las variables mas relevantes
rfgrid_rf2 <- expand.grid(mtry = c(3, 4, 5, 6))
modellist <- list()
#train with different ntree parameters
for (ntree in c(300,500,1000,1500,2000,2500)){
  set.seed(12345)
  fit <- train(
    factor(varObjBin) ~ h10pix + temp90 + trees + trees90 + Ymin + Ymax +
    temp + humid + humid90,
    data = dengue,
    method = "rf",
    trControl = control_rf,
    tuneGrid = rfgrid_rf2,
    lineout = FALSE,
    ntree = ntree,
    nodesize = 10,
    replace = TRUE,
    importance = TRUE)
  key <- toString(ntree)
  modellist[[key]] <- fit
}

results <- resamples(modellist)
dotplot(results)

dotplot(results)
```



fit\$bestTune

```
mtry
1 3
```

El gráfico dotplot() muestra el accuracy correspondiente a los trees evaluados en el ciclo for anterior, con esto los resultados en accuracy son similares pero por complejidad se prefiere dejar para pruebas en 500 trees. Para el parámetro *bestTune* el mtry es de 4 que también se dejara fijado en la validación cruzada.

5.3.2 Validacion cruzada para random forest

Se genera la validación cruzada para random forest utilizando la función *cruzadarfbn* y los parámetros obtenidos del estudio anterior, con *mtry*= 4 y *ntree* = 500

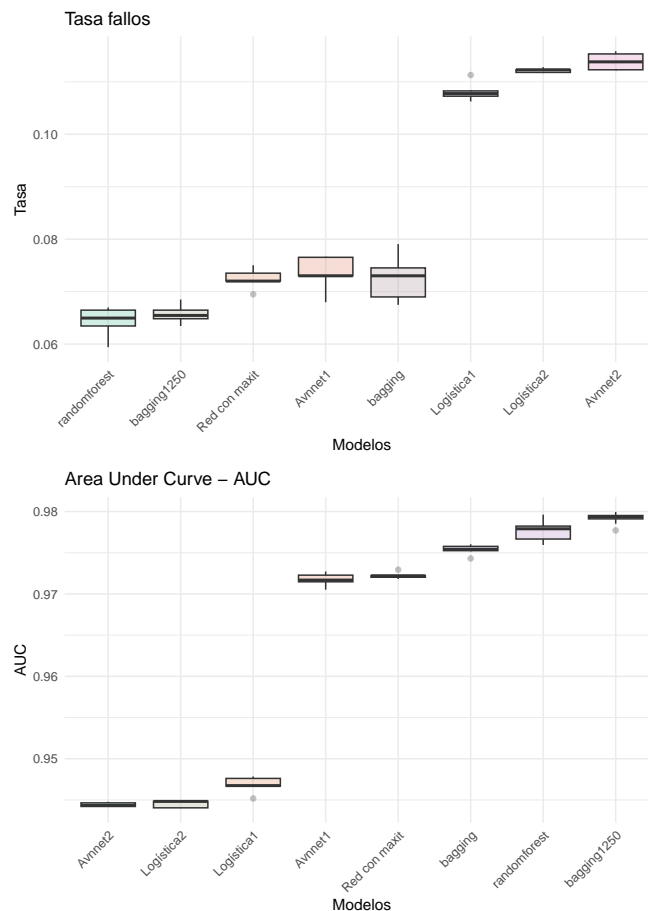
```
# Probar a dejar el parametro sampsize sin valor para que lo haga x defecto
medias11 <- cruzadarfbn(data = dengue, vardep = "varObjBin",
  listconti = c("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax",
    "temp", "humid", "humid90"),
  listclass = c(""),
  grupos = 4,
  sinicio = 1234,
  repe = 10,
  nodesize = 10,
  mtry = 4, ntree = 500, replace = TRUE)

medias11$modelo <- "randomforest"
```

5.3.3 Comparacion de medias y boxplot

Utilizando la función *genera_gráficos()* se generan los gráficos para realizar comparación de medias y evaluar los modelos

```
# 3. Compara medias -----
genera_graficos(medias1, medias2, medias3, medias4, medias5,medias6, medias9,medias11)
```



Observaciones:

- De los algoritmos evaluados hasta ahora, randomforest y bagging son los que muestran mejor rendimiento en AUC y tasa de fallos, dejando atrás las regresiones logísticas y las redes neuronales.

5.4 Stochastic Gradient Boosting

5.4.1 Tuning de gradient boosting

Se realiza un primer tuning del algoritmo gradient boosting con caret, probando los parámetros básicos de *shrinkage*, *n.minobsinnode* y *n.trees* mas las variables seleccionadas con *stepwise*

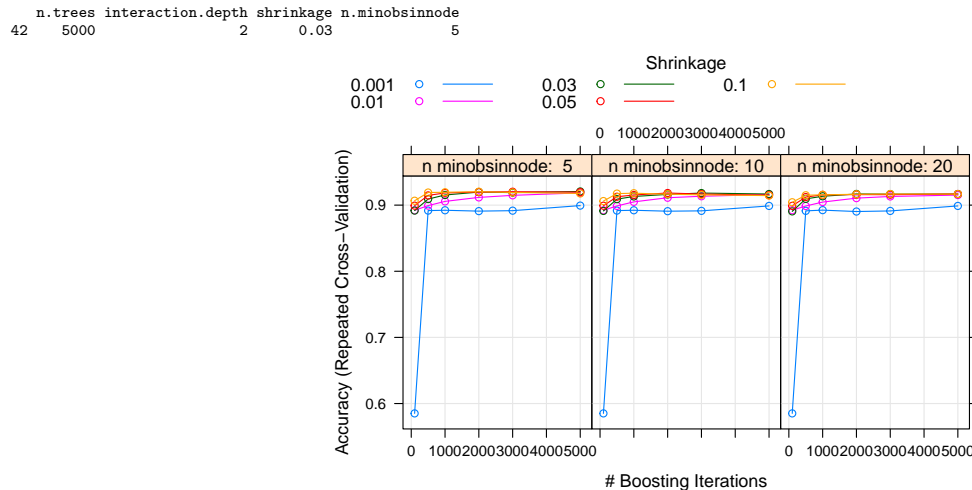
```
set.seed(12345)
# El shrinkage va desde los valores 0.0001 y 0.2
# Cuanto mas alto mas rapido pero puede resultar menos fino
# n.minobsnode mide la complejidad
# interaction.depth = c(2) arboles binarios

gb_grid <- expand_grid(
  shrinkage = c(0.1, 0.05, 0.03, 0.01, 0.001),
  n.minobsinnode = c(5,10,20),
  n.trees = c(100, 500, 1000, 2000, 3000, 5000),
  interaction.depth = c(2))

control_gb <- trainControl(method = "repeatedcv", number = 4, repeats = 5,
  savePredictions = "all", classProbs = TRUE)

gbm <- train(
  factor(varObjBin) ~ h10pix + temp90 + trees + trees90 + Ymin + Ymax + temp +
  humid + humid90,
  data = dengue,
  method = "gbm",
  trControl = control_gb,
  tuneGrid = gb_grid,
  distribution = "bernoulli",
  bag.fraction = 1,
  verbose = FALSE)

gbm
plot(gbm)
gbm$bestTune
```



Observaciones:

- Según gbm\$bestTune los mejores resultados se consiguen con 1000 iteraciones, shrinkage de 0.1 y n.minobsinnode de 5. Los graficos muestran que en iteraciones desde 500 en adelante no se observan diferencias considerables y tienen a compartir resultados especialmente desde el shrinkage 0.06. El máximo accuracy estaría entre 0.06 y 0.10 de shrinkage ademas de minobsinnode 5 tal como lo indica bestTune.

5.4.1.1 Estudio de early stopping Se prueba con algunos parámetros para buscar en que iteración se estabiliza el algoritmo

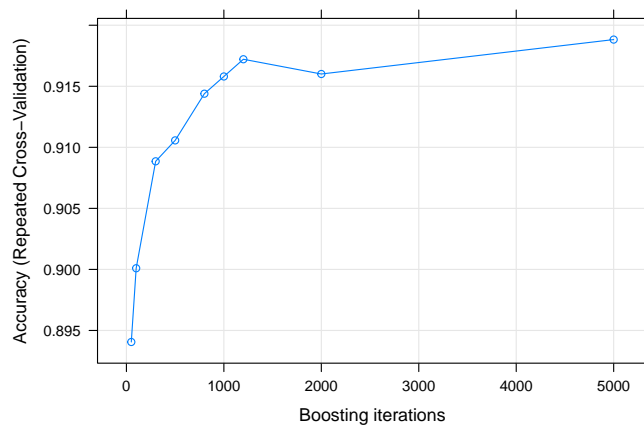
```
# Tuneado early stopping, se cambia bag.fraction por 0.5 en vez de 1
gbmgrid <- expand.grid(
  shrinkage = c(0.05),
  n.minobsinnode = c(5),
  n.trees = c(50, 100, 300, 500, 800, 1000, 1200, 2000, 5000),
  interaction.depth = c(2))

gbm2 <- train(
  factor(varObjBin) ~h10pix + temp90 + trees + trees90 + Ymin + Ymax + temp +
    humid + humid90,
  data = dengue,
  method = "gbm",
  trControl = control_gb,
  tuneGrid = gbmgrid,
  distribution = "bernoulli",
  bag.fraction = 0.5,
  verbose = FALSE)

gbm2$bestTune
plot(gbm2, xlab = "Boosting iterations")

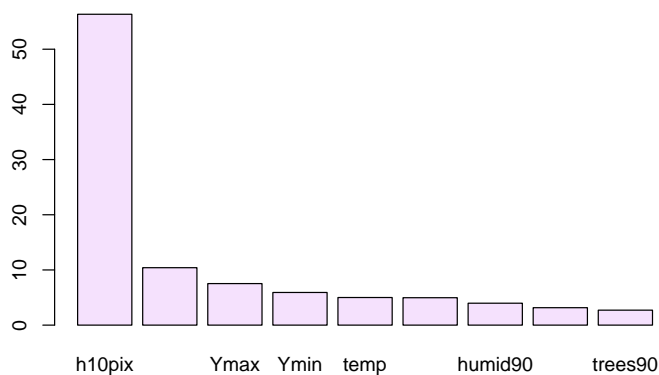
# Importancia variables
# summary(gbm2)
tabla <- summary(gbm2)
par(cex=1.5, las =2)
barplot(tabla$rel.inf, names.arg = row.names(tabla), col = "#F5E1FD",
  main = "Importancia de variables - gradient boosting")
```

```
n.trees interaction.depth shrinkage n.minobsinnode
9 5000 2 0.05 5
```



Importancia de variables

Importancia de variables – gradient boosting



Observaciones:

- Realizando el estudio de early stopping con las variables habituales ya seleccionadas y revisando los graficos, entre 1000 y 2000 n.trees comienza a estabilizarse el algoritmo, presentando resultados similares entre 2000 y 5000. Como se privilegia la simplicidad en el próximo apartado se realizan pruebas con n.trees de 2000.

5.4.2 Validacion cruzada para gradient boosting

Se genera la validación cruzada para gradient boosting utilizando los parámetros obtenidos en el proceso anterior de tuning:

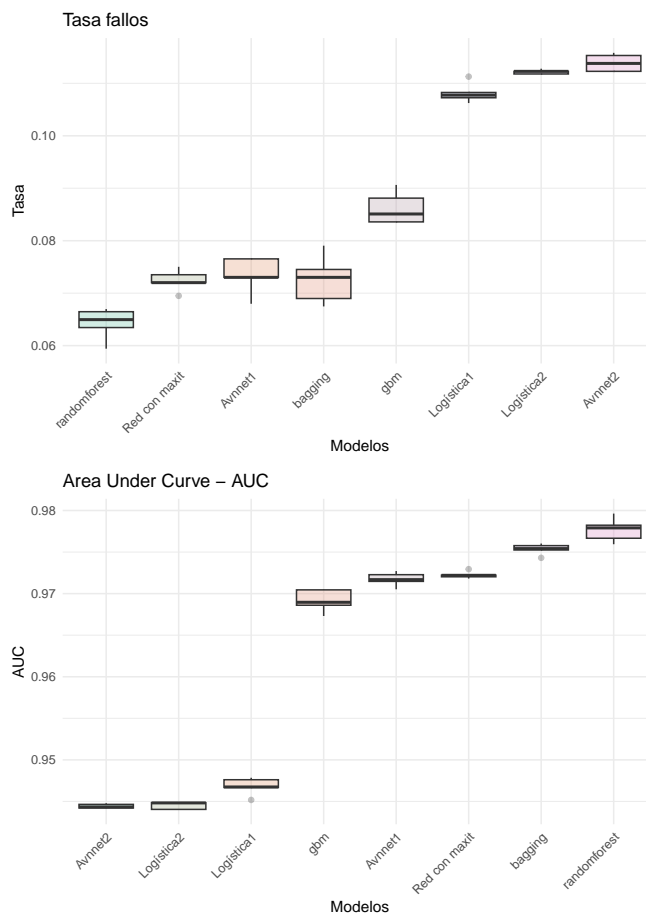
```
# 2. Validacion cruzada repetida gradient -----
# Segun el grafico entre 2000 y 5000 no hay tanta diferencia, se deja en 3000
medias12 <- cruzadagmbin(
  data = dengue,
  vardep="varObjBin",
  listconti=c("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax",
    "temp", "humid", "humid90"),
  listclass = c(""),
  grupos = 4,
  inicio = 1234,
  repe = 5,
  n.minobsinnode = 5,
  shrinkage = 0.05,
  n.trees = 2000,
  interaction.depth = 2)

medias12$modelo <- "gbm"
```

5.4.3 Comparacion de medias y boxplot

Utilizando la función `genera_gráficos()` se generan los gráficos para realizar comparación de medias y evaluar los modelos

```
# 2.1 Compara medias -----
genera_gráficos(medias1, medias2, medias3, medias4, medias5, medias6, medias11,
  medias12)
```



Observaciones:

- `gbm()` tiene un buen auc y esta en el medio del gráfico de la tasa de fallos, pero se observa mayor varianza comparándolo con otros modelos y no alcanza los niveles de accuracy que muestra un random forest o un bagging para este dataset.

5.5 Xgboost

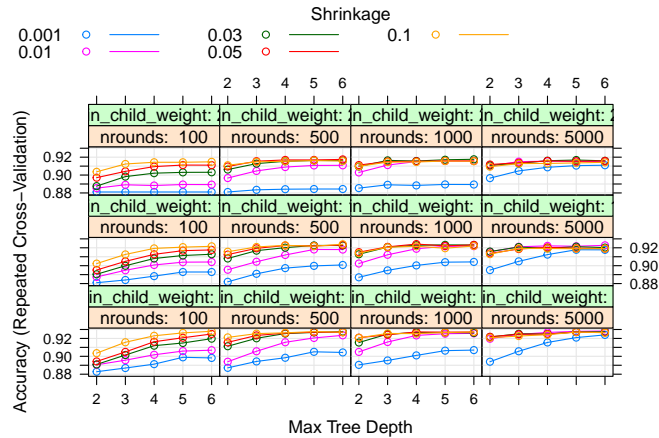
5.5.1 Tuning de Xgboost

```
## 3. Tuneo Xgboost -----
set.seed(12345)
xgbmgrid <- expand.grid(
  min_child_weight = c(5, 10, 20),
  eta = c(0.1, 0.05, 0.03, 0.01, 0.001),
  nrounds = c(100, 500, 1000, 5000),
  max_depth = c(2, 3, 4, 5, 6),
  gamma = 0,
  colsample_bytree = 1,
  subsample = 1)

control_xgboost <- trainControl(method = "repeatedcv", number = 4, repeats = 5,
  savePredictions = "all", classProbs = TRUE)

xgbm <- train(
  factor(varObjBin) ~h10pix + temp90 + trees + trees90 + Ymin + Ymax + temp +
  humid + humid90,
  data = dengue,
  method = "xgbTree",
  trControl = control_xgboost,
  tuneGrid = xgbmgrid,
  verbose = FALSE,
  verbosity = 0)

xgbm
plot(xgbm)
xgbm$bestTune
```



	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
112	5000	6	0.01	0	1	5	1

Observaciones:

- En la tabla anterior indica los que podrían ser mejores parámetros para el modelo, tomando en cuenta esta tabla el se modela el estudio de *early stopping* para confirmar lo indicado en tabla respecto al numero de interacciones sugeridas.

```
# Early stopping
# Con los parametros entregados anteriormente en best tune
# Con las variables ya preseleccionadas
xgbmgrid2 <- expand.grid(
  eta = c(0.1),
  min_child_weight = c(20),
  nrounds = c(50, 100, 150, 200, 250, 300, 500, 1000),
  max_depth = 6,
  gamma = 0,
  colsample_bytree = 1,
  subsample = 1)

set.seed(12345)
control_xgboost1 <- trainControl(method = "repeatedcv", number = 4, repeats = 5,
  savePredictions = "all", classProbs = TRUE)

xgbm2 <- train(
  factor(varObjBin) ~h10pix + temp90 + trees + trees90 + Ymin + Ymax + temp +
  humid + humid90,
```

```

data = dengue,
method = "xgbTree",
trControl = control_xgboost1,
tuneGrid = xgbmgrid2,
verbose = FALSE)

plot(xgbm2)

# Se cambia la semilla para observar variaciones
set.seed(45673)
control_xgboost2 <- trainControl(method = "repeatedcv", number = 4, repeats = 5,
savePredictions = "all", classProbs = TRUE)

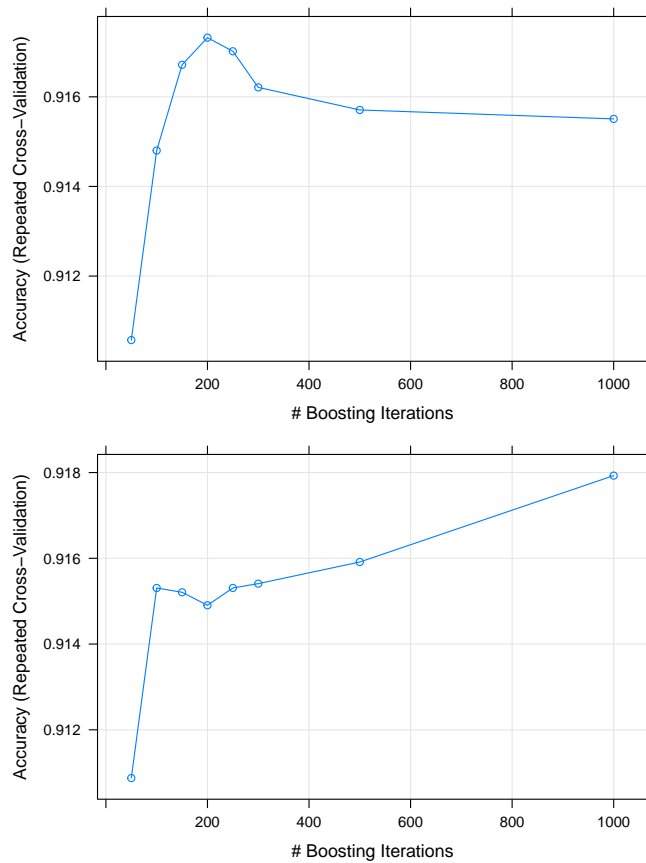
xgbm3 <- train(
  factor(var0bjBin) ~h10pix + temp90 + trees + trees90 + Ymin + Ymax + temp +
  humid + humid90,
  data = dengue,
  method = "xgbTree",
  trControl = control_xgboost2,
  tuneGrid = xgbmgrid2,
  verbose=FALSE)

xgbm3$bestTune
plot(xgbm3)

# Importancia de variables
plot(xgbm3)

```

5.5.1.1 Estudio de early stopping con xgboost



	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
4	200	6	0.1	0	1	20	1

	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
8	1000	6	0.1	0	1	20	1

```

# Solo con las variables seleccionadas en stepwise
xgbmgrid3 <- expand.grid(
  eta = c(0.1, 0.05, 0.03, 0.01, 0.001),
  min_child_weight = c(5, 10, 20),

```

```

nrounds = c(50,100,150,200,250,300),
max_depth = c(2,3,4,5,6),
gamma = 0,
colsample_bytree = 1,
subsample = 1)

set.seed(12345)
xgbm4 <- train(
  factor(varObjBin) ~ h10pix + h10pix90 + Xmax + humid90 +
  Ymax + temp90 + Xmin,
  data = dengue,
  method = "xgbTree",
  trControl = control_xgboost,
  tuneGrid = xgbmgrid3,
  verbose = FALSE)

xgbm4$bestTune
plot(xgbm4)

```

Observaciones

- En el código anterior se probó con una semilla diferente para ver si hay cambios en las interacciones. Aunque el accuracy es similar en ambos casos en su punto mas alto, con una semilla indica un numero de interacciones menor a lo recomendado en el primer tuning y con semilla distinta a la usual alcanza la estabilidad a partir de 1000 boosting iterations.

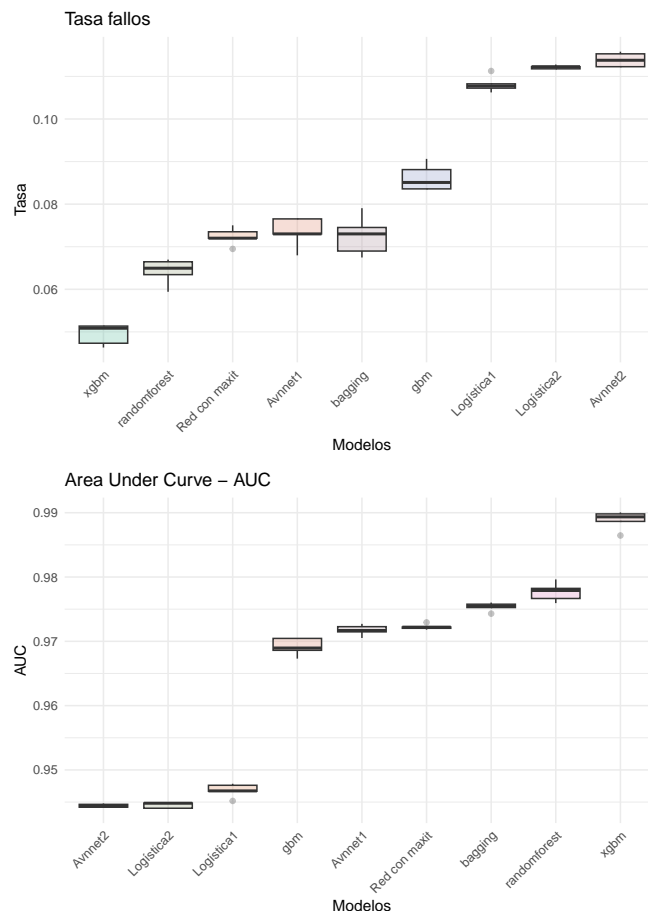
5.5.2 Comparacion de medias y boxplot

Utilizando la función `genera_gráficos()` se generan los gráficos para realizar comparación de medias y evaluar los modelos

```

genera_graficos(medias1, medias2, medias3, medias4, medias5,medias6, medias11,
  medias12, medias13)

```



Observaciones: - Xgbm alcanza resultados muy buenos en relación a auc y tasa de fallos, inclusive con varianza mas pequeña, demostrando la popularidad con la que cuenta el algoritmo, quedando a la cabeza del gráfico mostrado. Como punto a considerar, computacionalmente es mas costoso que un random forest o bagging que también presenta muy buenos resultados por lo que se debe tener este punto en cuenta al momento de elegir modelo ganador.

5.6 Support Vector Machines

5.6.1 Tuning de support vector machines o SVM

5.6.1.1 SVM Lineal Para el SVM lineal se necesita la constante de regularización C. A continuación se construyen un par de modelos buscando un valor C óptimo utilizando las variables seleccionadas con *stepwise*:

Como en los modelos anteriores, se utiliza el archivo dengue.RDS que contiene los datos preprocesados

```
# Carga de datos
dengue <- readRDS(file = "./data/processed/dengue.rds")

# 1. SVM -----
# 1.1 SVM lineal ----

SVMgrid <- expand.grid(C = c(0.01, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10))
set.seed(12345)
control_svm <- trainControl(method = "repeatedcv", number = 4, repeats = 5,
                             savePredictions = "all", classProbs = TRUE)

svm_1 <- train(
  data = dengue,
  factor(varObjBin) ~ h10pix + temp90 + trees + trees90 + Ymin +
  Ymax + temp + humid + humid90,
  method = "svmLinear",
  trControl = control_svm,
  tuneGrid = SVMgrid,
  verbose = FALSE)
```

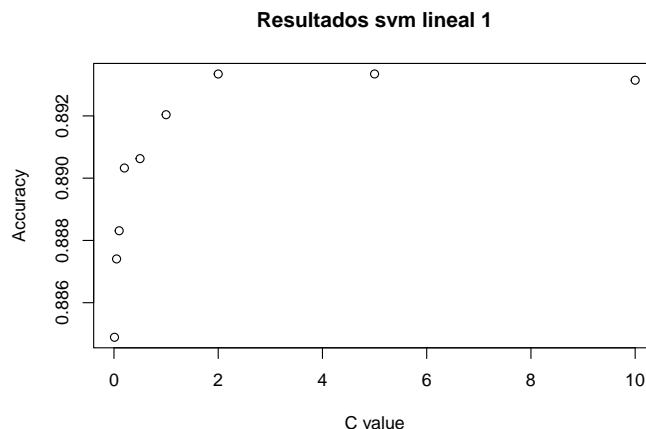
Se muestran los resultados y el gráfico correspondiente

```
#svm_1
knitr::kable(svm_1$results, "pipe")
```

C	Accuracy	Kappa	AccuracySD	KappaSD
0.01	0.8848896	0.7657148	0.0135121	0.0268383
0.05	0.8874055	0.7717453	0.0143168	0.0282380
0.10	0.8883113	0.7737057	0.0149961	0.0296287
0.20	0.8903258	0.7779425	0.0148499	0.0294770
0.50	0.8906274	0.7786645	0.0154525	0.0307425
1.00	0.8920387	0.7814860	0.0149960	0.0298476
2.00	0.8933464	0.7842393	0.0151495	0.0302194
5.00	0.8933472	0.7842325	0.0155040	0.0310005
10.00	0.8931454	0.7837290	0.0162712	0.0325210

```
svm_1$bestTune
```

```
C
8 5
plot(svm_1$results$C, svm_1$results$Accuracy, xlab = "C value", ylab = "Accuracy",
     main = "Resultados svm lineal 1")
```



Observando `bestTune` y el gráfico, para este primer modelado de svm el valor C mas adecuado estaría entre 0.20 y 2 (donde se observa mejor *accuracy* en tabla), caret en `bestTune` indica un valor de C de 1 que es también lo que se observa en el gráfico. Con este nuevo valor de C se construye otro modelo donde se busca si es que hay un mejor valor de C en un grid de 0.1 hasta 1

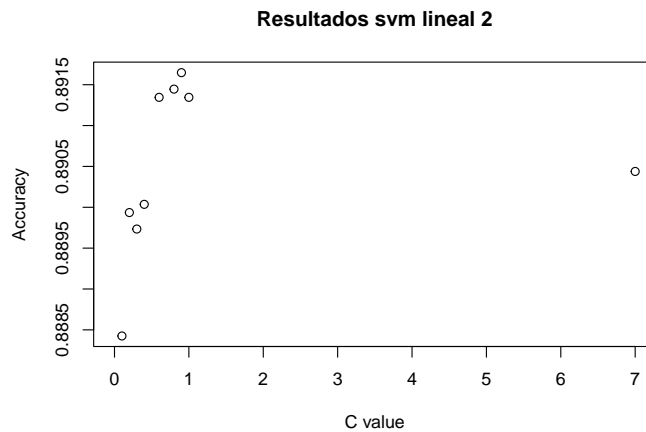
```
SVMgrid_1 <- expand.grid(C = c(0.1, 0.2, 0.3, 0.4, 0.6, 0.7, 0.8, 0.9, 1))

svm_2 <- train(
  data = dengue,
  factor(varObjBin) ~ h10pix + temp90 + trees + trees90 + Ymin +
  Ymax + temp + humid + humid90,
  method = "svmLinear",
  trControl = control_svm,
  tuneGrid = SVMgrid_1,
  verbose = FALSE)
```

Se despliegan los resultados

C	Accuracy	Kappa	AccuracySD	KappaSD
0.0	NaN	NaN	NA	NA
0.1	0.8884251	0.7740103	0.0172724	0.0341358
0.2	0.8899354	0.7772673	0.0169244	0.0334794
0.3	0.8897338	0.7768704	0.0173686	0.0343322
0.4	0.8900360	0.7775201	0.0170877	0.0337618
0.6	0.8913452	0.7801687	0.0176056	0.0349052
0.8	0.8914459	0.7804207	0.0182903	0.0362346
0.9	0.8916475	0.7808622	0.0179788	0.0355690
1.0	0.8913452	0.7802015	0.0184476	0.0365251
7.0	0.8904384	0.7782801	0.0171994	0.0340750

C
8 0.9



Para este dataset con SVM lineal el valor de C mas adecuado seria C=1.

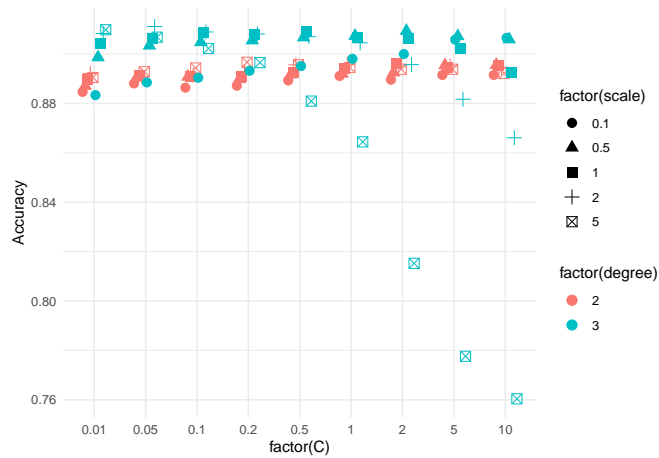
5.6.1.2 SVM Polinomial En este segundo apartado se utiliza un kernel polinomial buscando el valor de C, el grado del polinomio y la escala (C, degree y scale respectivamente), se entrena el modelo:

```
# 1.2 SVM Polinomial ----
SVMgrid_p <- expand.grid(
  C = c(0.01, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10),
  degree = c(2, 3),
  scale = c(0.1, 0.5, 1, 2, 5))

svm_3 <- train(
  data = dengue,
  factor(varObjBin) ~ h10pix + temp90 + trees + trees90 +
  Ymin + Ymax + temp + humid + humid90,
  method = "svmPoly",
  trControl = control_svm,
  tuneGrid = SVMgrid_p,
  verbose = FALSE)
```

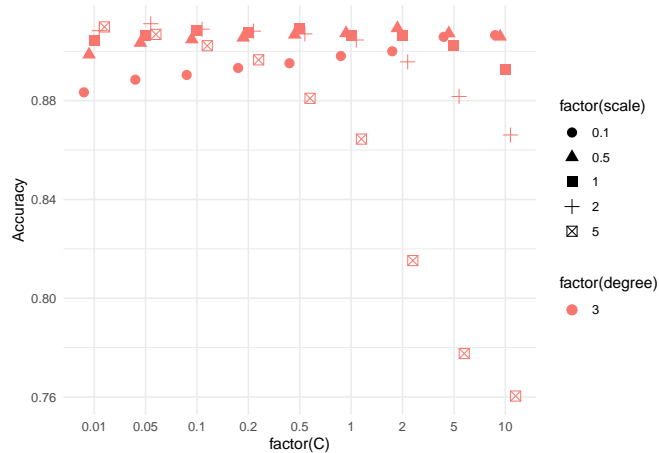
Se despliegan los resultados

degree	scale	C
19	3	2 0.05



En los resultados con bestTune se sugiere un degree de 3, scale 2 y un c value de 0.5. En el gráfico los mejores resultados en *accuracy* los muestra con degree 3, para c value muestra buenos resultados de 0.5 en adelante.

Como los mejores resultados los da el degree 3, se muestra un nuevo gráfico con ese grado y realizar observaciones:



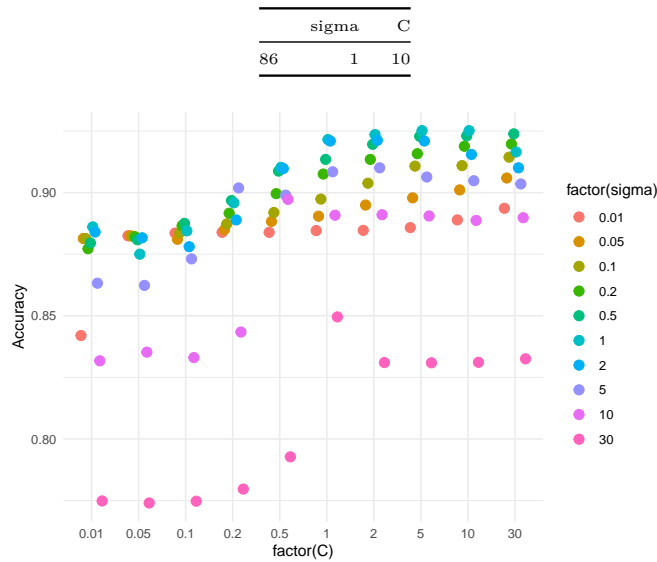
En general scale tiende a generar una curva hasta el valor $C=0.5$ y después descender para volver a alcanzar un accuracy alto en $C=10$. Como un C muy grande puede tender al sobreajuste para este dataset es mejor un C 0,5 o 1 y escala 2

5.6.1.3 SVM RBF En este tercer apartado se agrega al c value el parámetro sigma de varianza-escala buscando el valor gamma/sigma mas adecuado para este modelo. Gamma: a mayor gamma se puede presentar sobreajuste.

```
# 1.3 SVM RBF -----
#Se agrega el parametro sigma
SVMgrid_rbf <- expand.grid(
  C = c(0.01, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 30),
  sigma = c(0.01, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 30)
)

svm_4 <- train(
  data = dengue, factor(varObjBin) ~ h10pix + temp90 + trees + trees90 +
  Ymin + Ymax + temp + humid + humid90,
  method = "svmRadial",
  trControl = control_svm,
  tuneGrid = SVMgrid_rbf,
  verbose = FALSE)
```

Se despliegan los resultados



- A mayor sigma (10, 30) no se observa que los resultados mejoren en términos de *accuracy*.
- bestTune recomienda un sigma de 0.5 y un C de 30, pero desde c 1 hasta c 30 los resultados se observan relativamente parejos por lo que se mantiene el dejar el valor c en 1 y un sigma de 0.5

5.6.2 Validacion cruzada para support vector machines

Con los valores obtenidos de svm lineal, polinomial y RBF se realiza la validación cruzada para su posterior ploteo y comparación de medias para este modelo versus los generados con los algoritmos anteriores

```
# 2. Validacion cruzada rep SVM -----
#cv para lineal
medias14 <- cruzadaSVMbin(
  data = dengue,
  vardep = "varObjBin",
  listconti = c("h10pix", "temp90", "trees", "trees90",
    "Ymin", "Ymax", "temp", "humid", "humid90"),
  listclass = c(""),
  grupos = 4,
  sinicio = 1234,
  repe = 5,
  C = 1
)

medias14$modelo <- "svmLineal"

# cv para poli
medias15 <- cruzadaSVMbinPoly(
  data = dengue,
  vardep = "varObjBin",
  listconti = c("h10pix", "temp90", "trees", "trees90",
    "Ymin", "Ymax", "temp", "humid", "humid90"),
  listclass = c(""),
  grupos = 4,
  sinicio = 1234,
  repe = 5,
  C = 0.5,
  degree = 3,
  scale = 2
)

medias15$modelo <- "svmPoly"

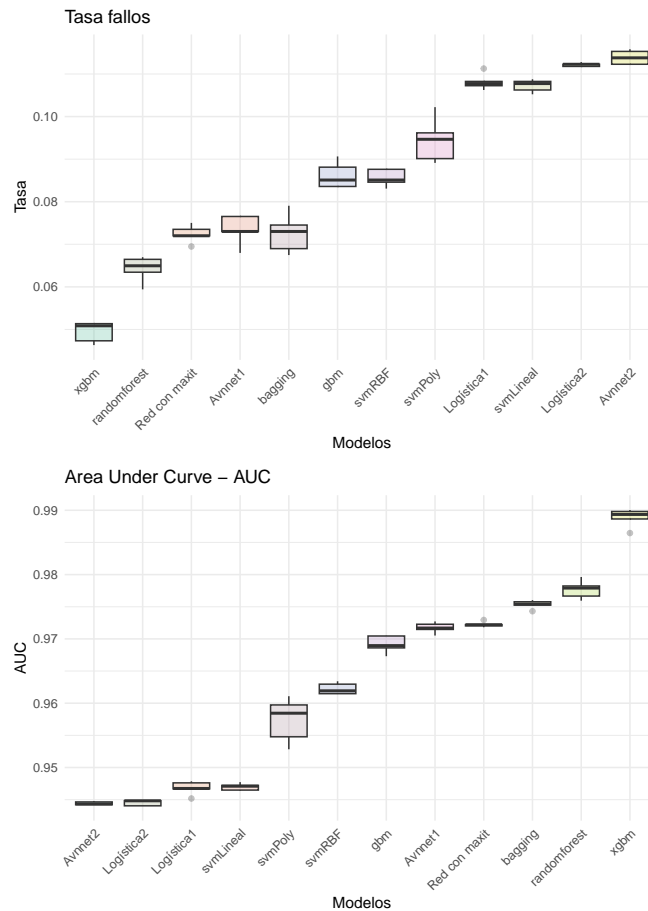
# cv RBF
medias16 <- cruzadaSVMbinRBF(
  data = dengue,
  vardep = "varObjBin",
  listconti = c("h10pix", "temp90", "trees", "trees90",
    "Ymin", "Ymax", "temp", "humid", "humid90"),
  listclass = c(""),
  grupos = 4,
  sinicio = 1234,
  repe = 5,
  C = 1,
  sigma = 0.5
)

medias16$modelo <- "svmRBF"
```

5.6.3 Comparacion de medias y boxplot

Utilizando la función `genera_graficos()` se despliega el gráfico de comparación de medias

```
# 3. Compara medias ----
genera_graficos(medias1,medias2,medias3, medias4, medias5,medias6, medias11, medias12,
medias13, medias14,medias15, medias16)
```



Observaciones

- De los tres modelos el que presenta mayor varianza es el modelo polinomial, los resultados en accuracy sobre todo en svmRBF son buenos pero sigue siendo hasta ahora un randomForest o un xgbm (solo observando el gráfico) los que mejores resultados entregan.

6 Ensamblado de algoritmos

Nota: El código completo de la practica de ensamblados se encuentra en el siguiente repositorio de GitHub, para no extender innecesariamente el reporte:

https://github.com/TiaIvonne/Machine-LearningR/blob/master/5_Ensamblado.R

6.1 Preparacion del ensamblado con *caret ensemble*

Antes de comenzar el modelado, se deben tener decididos los parámetros para los algoritmos que serán ensamblados (proceso obtenido anteriormente en el apartado de tuning) para los siguientes algoritmos:

Redes neuronales Gradient boosting machine Random Forest Classifier Support Vector Machines (lineal, polinomial y radial)

Con los resultados obtenidos se construyen los *grids* y se realiza el modelamiento

```

# Semilla
set.seed(12345)
repeticiones <- 10

# Evaluar los modelos
stackControl <- trainControl(
  method = "repeatedcv",
  number = 4,
  repeats = repeticiones,
  savePredictions = TRUE,
  classProbs = TRUE)

# grid gbm
gbmGrid <- expand.grid(
  n.trees = c(2000),
  interaction.depth = c(2),
  shrinkage = c(0.1),
  n.minobsinnode = c(20))

# grid random forest
rfGrid <- expand.grid(mtry=c(3))

# grid svm lineal
svmlinGrid <- expand.grid(C=c(0.08))

# grid svm ply
svmPolyGrid <- expand.grid(C=c(0.03), degree=c(3), scale=c(2))

# grid svm radial
svmRadialGrid <- expand.grid(sigma = c(0.5), C = c(30))

# Modelado
set.seed(12345)
models <- caretList(varObjBin~,
  data = dengue,
  trControl = stackControl,
  tuneList = list(
    parrrf = caretModelSpec(method = "rf", maxnodes = 30, n.trees = 200, nodesize = 10, sampsize = 150, tuneGrid = rfGrid),
    glm = caretModelSpec(method = "glm"),
    gbm = caretModelSpec(method = "gbm", tuneGrid = gbmGrid),
    svmLinear = caretModelSpec(method = "svmLinear", tuneGrid = svmlinGrid),
    svmPoly = caretModelSpec(method = "svmPoly", tuneGrid = svmPolyGrid),
    svmradial = caretModelSpec(method = "svmRadial", tuneGrid = svmRadialGrid)))

```

Se despliegan los resultados obtenidos para el ensamblado

```

results <- resamples(models)
summary(results)
dotplot(results)

modelCor(results)
splom(results)
results[[2]]

ensemble <- caretEnsemble(models)
# Aquí se recomiendan los pesos para el ensamblado # de todos los modelos y se
# ve la tasa de aciertos de cada modelo y ensamblado

summary(ensemble)

# 2. Validacion cruzada y kit ensamblado ----
# 2.1 Leer funciones -----
# 2.2 Preparar archivo, variables , semilla y repeticiones ----
dput(names(dengue))

set.seed(12345)

archivo <- dengue

vardep <- "varObjBin"
listconti <- c("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax",
  "temp", "humid", "humid90")

listclass <- c("")
grupos <- 4
inicio <- 1234
repe <- 15

# 2.3 Obtener datos de cv repetida para cada algoritmo y
# procesar resultado ----
medias_1<-cruzadalogistica(data = archivo,
  vardep = vardep,
  listconti=listconti,
  listclass = listclass, grupos = grupos,
  inicio = inicio, repe = repe)

medias1bis <- as.data.frame(medias_1[1])
medias1bis$modelo<-"logistica"
predi1<-as.data.frame(medias_1[2])
predi1$logi<-predi1$Yes

medias_2<-cruzadaavnetbin(data=archivo,
  vardep=vardep,listconti=listconti,
  listclass=listclass,grupos=grupos,sinicio=inicio,repe=repe,

```

```

size=c(10),decay=c(0.01),repeticiones=5,itera=200)

medias2bis<-as.data.frame(medias_2[1])
medias2bis$modelo<-"avnnnet"
predi2<-as.data.frame(medias_2[2])
predi2$avnnnet<-predi2$Yes

medias_3<-cruzadarfbin(data=archivo,
  vardep=vardep,listconti=listconti,
  listclass=listclass,grupos=grupos,sinicio=sinicio,repe=repe,
  mtry=3,ntree=500,nodesize=10,replace=TRUE)

medias3bis<-as.data.frame(medias_3[1])
medias3bis$modelo<-"randomforest"
predi3<-as.data.frame(medias_3[2])
predi3$rf<-predi3$Yes

medias_4 <-cruzadagbmbin(data=archivo,
  vardep=vardep,listconti=listconti,
  listclass=listclass,grupos=grupos,sinicio=sinicio,repe=repe,
  n.minobsinnode=5,shrinkage=0.1,n.trees=3000,interaction.depth=2)

medias4bis<-as.data.frame(medias_4[1])
medias4bis$modelo<-"gbm"
predi4<-as.data.frame(medias_4[2])
predi4$gbm<-predi4$Yes

medias_5 <-cruzadaxgbmbin(data = archivo,
  vardep = vardep,listconti = listconti,
  listclass = listclass, grupos = grupos, inicio = inicio, repe = repe,
  min_child_weight = 5, eta = 0.10, nrounds = 250, max_depth = 6,
  gamma = 0, colsample_bytree = 1, subsample = 1,
  alpha = 0, lambda = 0, lambda_bias = 0)

medias5bis <-as.data.frame(medias_5[1])
medias5bis$modelo <- "xgbm"
predi5 <-as.data.frame(medias_5[2])
predi5$xgbm<-predi5$Yes

medias_6<-cruzadaSVMbin(data=archivo,
  vardep=vardep,listconti=listconti,
  listclass=listclass,grupos=grupos,
  inicio=sinicio,repe=repe,C=0.08)

medias6bis<-as.data.frame(medias_6[1])
medias6bis$modelo<- "svmLinear"
predi6<-as.data.frame(medias_6[2])
predi6$svmLinear<-predi6$Yes

medias_7 <- cruzadaSVMbinPoly(data = archivo,
  vardep = vardep, listconti = listconti,
  listclass = listclass, grupos = grupos, inicio = inicio, repe = repe,
  C = 0.03, degree = 3, scale = 2)

medias7bis <- as.data.frame(medias_7[1])
medias7bis$modelo <- "svmPoly"
predi7 <- as.data.frame(medias_7[2])
predi7$svmPoly <- predi7$Yes

medias_8 <- cruzadaSVMbinRBF(data = archivo,
  vardep = vardep, listconti = listconti,
  listclass = listclass, grupos = grupos,
  inicio = inicio, repe = repe,
  C = 30, sigma = 0.5)

medias8bis<-as.data.frame(medias_8[1])
medias8bis$modelo<- "svmRadial"
predi8<-as.data.frame(medias_8[2])
predi8$svmRadial<-predi8$Yes

summary(ensemble)

```

```

The following models were ensemble: parrrf, glm, gbm, svmlinear, svmPoly, svmradial
They were weighted:
4.1057 -0.3249 0.4592 -2.7462 -1.4311 -1.5155 -3.104
The resulting Accuracy is: 0.9457
The fit for each individual model on the Accuracy is:
  method Accuracy AccuracySD
  parrrf 0.8973276 0.013143839
    glm 0.8899252 0.012870074
    gbm 0.9378647 0.009223015
svmlinear 0.8885664 0.011650506
  svmPoly 0.9256787 0.011685866
svmradial 0.9385687 0.009063826

```

Observaciones

6.2 Validacion cruzada repetida y boxplot

Siguiendo la guia de ensamblados entregada en el material, se realizan los pasos requeridos para realizar pruebas de ensamblado y validacion cruzada repetida

6.2.1 Paso 1, carga del archivo con la funcion “cruzadas ensamblado binaria fuente.R”

6.2.2 Paso 2, Preparacion del archivo

Se define semilla, variables y repeticiones

6.2.3 Paso 3, aplicacion de la funcion cruzadas ensamblado

Con los datos obtenidos del proceso de tuning se construyen las nuevas medias a evaluar bajo la función “cruzadas ensamblado binaria fuente.R”.

Solo se adjunta una muestra del código generado

```
medias_1<-cruzadalogistica(data = archivo,
  vardep = vardep,
  listconti=listconti,
  listclass = listclass, grupos = grupos,
  inicio = inicio, repe = repe)

medias1bis <- as.data.frame(medias_1[1])
medias1bis$modelo<-"logistica"
predi1<-as.data.frame(medias_1[2])
predi1$logi<-predi1$Yes

medias_2<-cruzadaavnnnetbin(data=archivo,
  vardep=vardep,listconti=listconti,
  listclass=listclass,grupos=grupos,sinico=sinico,repe=repe,
  size=c(10),decay=c(0.01),repeticiones=5,itera=200)

medias2bis<-as.data.frame(medias_2[1])
medias2bis$modelo<-"avnnnet"
predi2<-as.data.frame(medias_2[2])
predi2$avnnnet<-predi2$Yes

medias_3<-cruzadarfbin(data=archivo,
  vardep=vardep,listconti=listconti,
  listclass=listclass,grupos=grupos,sinico=sinico,repe=repe,
  mtry=3,ntree=500,nodesize=10,replace=TRUE)

medias3bis<-as.data.frame(medias_3[1])
medias3bis$modelo<-"randomforest"
predi3<-as.data.frame(medias_3[2])
predi3$rfr<-predi3$Yes

# Se ha omitido el resto del código
```

Con la función genera graficos y una vez calculadas las medias en el proceso anterior se obtiene el gráfico de cajas respectivo:

```
# Genera graficos
genera_graficos(medias1bis, medias2bis, medias3bis, medias4bis, medias5bis,
  medias6bis, medias7bis, medias8bis)
```

6.2.4 Paso 4, construccion del ensamblado

Con las predicciones obtenidas en las respectivas variables predi1, predi2, predi3 etc se crean los ensamblados. Solo se adjunta una muestra del código

```
unipredi<-cbind(predi1,predi2,predi3,predi4,predi5,predi6,predi7,predi8)
ncol(unipredi)

unipredi<- unipredi[, !duplicated(colnames(unipredi))]
ncol(unipredi)

unipredi$predi9<-(unipredi$logi+unipredi$avnnnet)/2
unipredi$predi10<-(unipredi$logi+unipredi$rfr)/2
unipredi$predi11<-(unipredi$logi+unipredi$gbm)/2
unipredi$predi12<-(unipredi$logi+unipredi$xbgm)/2
unipredi$predi13<-(unipredi$logi+unipredi$svmLinear)/2
unipredi$predi14<-(unipredi$logi+unipredi$svmPoly)/2
```

6.2.5 Paso 5, Procesado de los ensamblados

Solo se adjunta una parte del código, se construyen los promedios de tasa de fallos y AUC.

```
dput(names(unipredi))
listado<-c("logi", "avnnnet",
  "rfr", "gbm", "xbgm", "svmLinear", "svmPoly",
  "svmRadial", "predi9", "predi10", "predi11", "predi12",
  "predi13", "predi14", "predi15", "predi16", "predi17", "predi18",
  "predi19", "predi20", "predi21", "predi22", "predi23", "predi24",
  "predi25", "predi26", "predi27", "predi28", "predi29", "predi30",
  "predi31", "predi32", "predi33", "predi34", "predi35", "predi36",
  "predi37", "predi38", "predi39", "predi40", "predi41", "predi42",
```



```

      "predi43", "predi44", "predi45", "predi46", "predi47", "predi48",
      "predi49", "predi50", "predi51", "predi52", "predi53", "predi54",
      "predi55", "predi56", "predi57", "predi58", "predi59", "predi60",
      "predi61", "predi62", "predi63", "predi64", "predi65", "predi66",
      "predi67", "predi68", "predi69")

# Cambio a Yes, No, todas las predicciones
# Defino funcion tasafallos

tasafallos<-function(x,y) {
  confu<-confusionMatrix(x,y)
  tasa<-confu[[3]][1]
  return(tasa)
}

auc<-function(x,y) {
  curvaroc<-roc(response=x,predictor=y)
  auc<-curvaroc$auc
  return(auc)
}

# Se obtiene el numero de repeticiones CV y se calculan las medias por repe en
# el data frame medias0

repeticiones<-nlevels(factor(unipredi$Rep))
unipredi$Rep<-as.factor(unipredi$Rep)
unipredi$Rep<-as.numeric(unipredi$Rep)

medias0<-data.frame(c())
for (prediccion in listado)
{
  unipredi$proba<-unipredi[,prediccion]
  unipredi[,prediccion]<-ifelse(unipredi[,prediccion]>0.5,"Yes","No")
  for (repe in 1:repeticiones)
  {
    paso <- unipredi[(unipredi$Rep==repe),]
    pre<-factor(paso[,prediccion])
    archi<-paso[,c("proba","obs")]
    archi<-archi[order(archi$proba),]
    obs<-paso[,c("obs")]
    tasa=1-tasafallos(pre,obs)
    t<-as.data.frame(tasa)
    t$modelo<-prediccion
    auc<-suppressMessages(auc(archi$obs,archi$proba))
    t$auc<-auc
    medias0<-rbind(medias0,t)
  }
}

```

6.2.6 Paso 6, boxplot inicial

Boxplot con tasa de fallos para todos los ensamblados, en el punto 8 se muestran ordenados

6.2.7 Paso 7, tabla con resultados

Se genera tabla con resultados para la tasa de fallos y el área under curve. **Nota:** Solo se despliegan las primeras salidas para no extender el documento.

```

tablamedias<-medias0 %>%
  group_by(modelo) %>%
  summarise(tasa=mean(tasa))

tablamedias<-as.data.frame(tablamedias[order(tablamedias$tasa),])
knitr::kable(head(tablamedias, n = 10), "pipe")

```

modelo	tasa
predi55	0.0641826
predi16	0.0650554
rf	0.0652904
predi57	0.0656260
predi26	0.0663310
predi60	0.0663646
predi63	0.0663981
predi69	0.0667338
predi52	0.0670695
predi56	0.0678751

```

# Para AUC
tablamedias2<-medias0 %>%
  group_by(modelo) %>%
  summarise(auc=mean(auc))

tablamedias2<-tablamedias2[order(-tablamedias2$auc),]
knitr::kable(head(tablamedias2, n=10), "pipe")

```

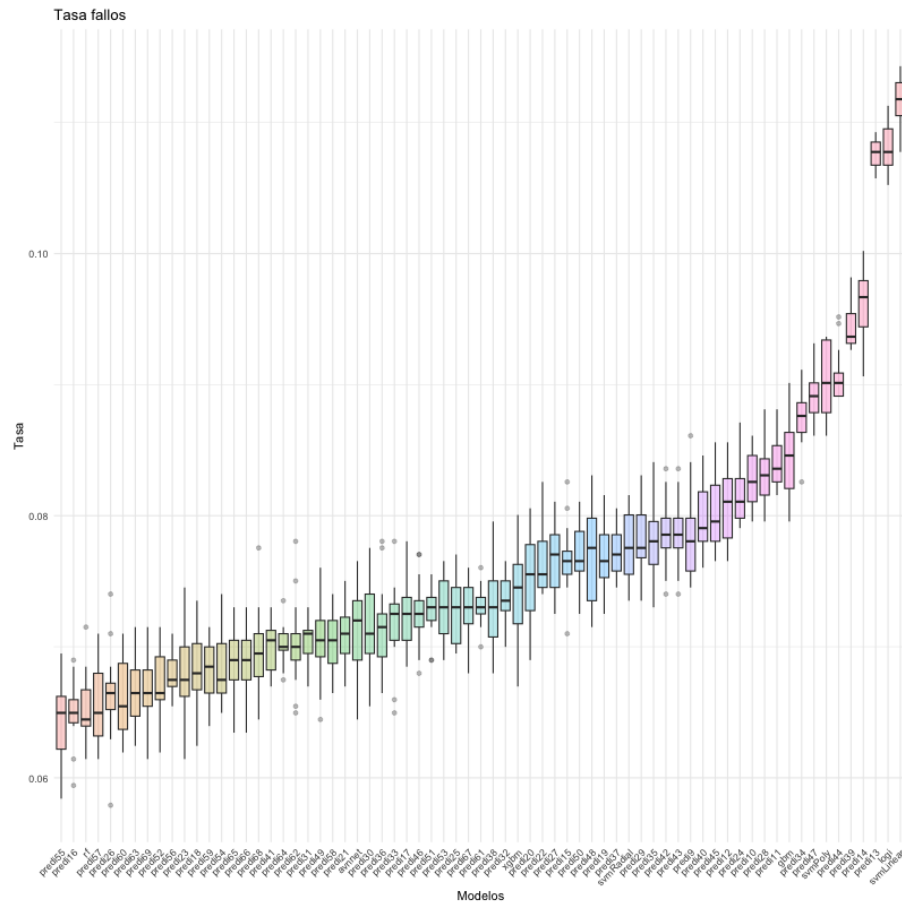
modelo	auc
predi55	0.9789378
predi57	0.9786039
predi52	0.9782842
predi60	0.9782593
predi23	0.9780586
predi69	0.9779774
predi56	0.9778908
predi26	0.9778800
predi16	0.9777369
predi54	0.9775005

En el punto siguiente se gráfica para una mejor visualización y toma de decisión

6.2.8 Paso 8, Boxplot ordenados para comparacion de medias

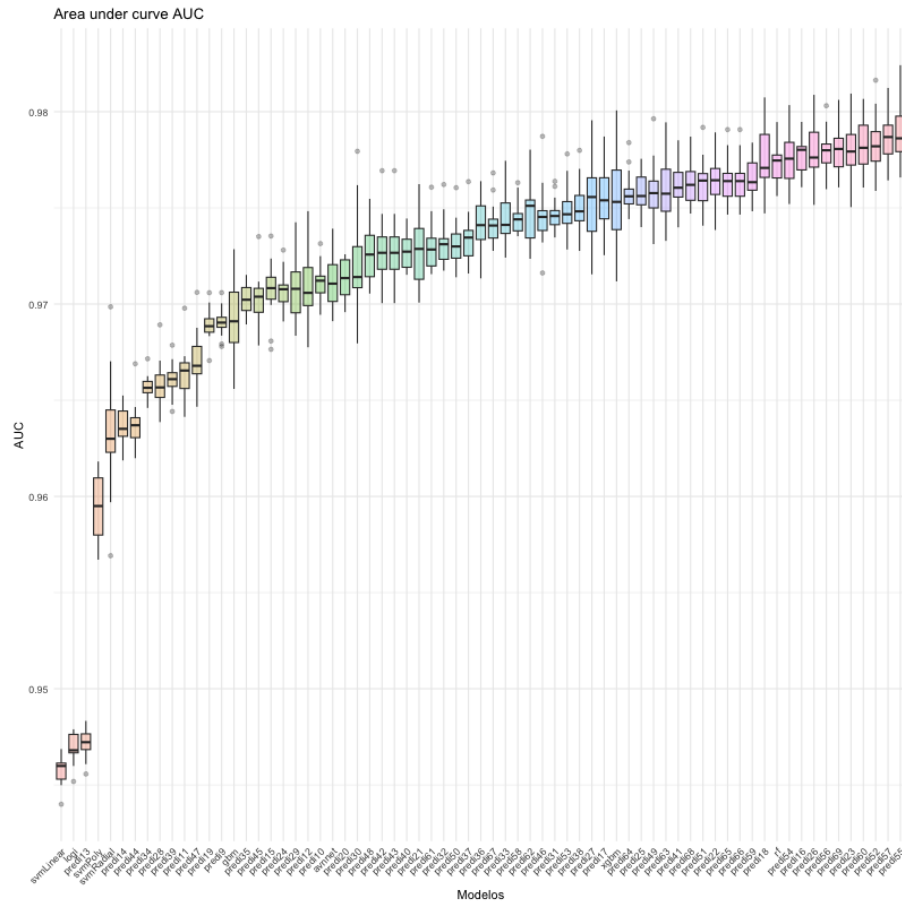
Con los resultados obtenidos de los ensamblados se generan los boxplot ordenados por tasa de fallos y auc respectivamente:

```
knitr::include_graphics("../figure/ensamblado_medias0_tasa.png")
```



En tasa de fallos los modelos con menor tasa son predi55, predi16, randomforest

```
knitr::include_graphics("../figure/ensamblado_medias0_auc.png")
```

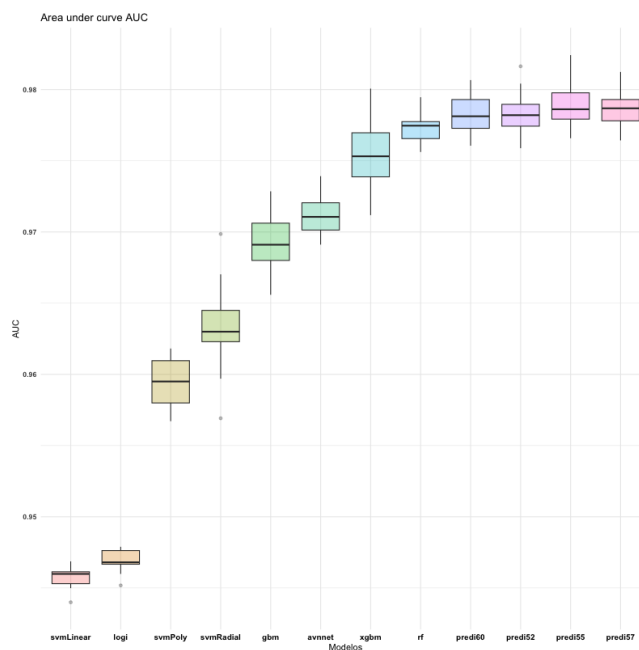


En AUC respectivamente los modelos con mayor *accuracy* son predi55, predi57 y predi52.

Como se observa en el gráfico son demasiados modelos, en el apartado siguiente solo se trafican los mejores modelos de ensamblado y se comparan con los algoritmos sin ensamblar

6.2.9 Paso 9, comparacion de mejores modelos

Los mejores ensamblados son los modelos predi57, predi55, predi52 y predi 60, estos se muestran a continuación comparándolos con los modelos originales sin ensamblar



6.2.10 Paso 10, revision a los mejores ensamblados

```
unipredi$predi55<-(unipredi$rf+unipredi$xgbm+unipredi$svmRadial)/3
unipredi$predi57<-(unipredi$rf+unipredi$avnnet+unipredi$xgbm)/3
unipredi$predi60<-(unipredi$rf+unipredi$avnnet+unipredi$svmRadial)/3
unipredi$predi52<-(unipredi$rf+unipredi$gbm+unipredi$svmRadial)/3
```

En los ensamblados el algoritmo común en los cuatro mejores ensamblados es random forest, el mejor ensamblado es el numero 55 que mezcla random forest, xgbm y svm radial.

6.2.11 Observaciones finales

- De los modelos originales los mejores resultados observando el gráfico los obtiene random forest, xgbm y avnnet, xgbm presenta mas varianza que los dos anteriores.
- El *accuracy* alcanzado por los ensamblados es mas alto si se compara con los modelos originales sin ensamblar. Si solo fuese tomando como criterio el accuracy, predi57 que es un ensamblado de randomforest con xgbm y svmlineal seria el modelo *ganador*. Sin embargo no hay diferencias dramáticas desde randomforest hacia adelante y considerando sesgo-varianza, randomforest estaría mostrando mejores resultados que los ensamblados.
- La tasa de fallos vs el auc muestra algunas incoherencias, en tasa de fallos el numero menor lo obtienen predi55 y predi16 pero en auc los ensamblados predi55 y predi57 respectivamente están a la cabecera de los resultados.

7 Evaluacion del modelo ganador y conclusiones

7.1 Algoritmo a usar para este modelo

Se ha decidido para esta practica que el algoritmo ganador es *random forest classifier*, los resultados son de los mejores observados en grafico, combinado con AUC y tasa de fallos, presentando resultados competitivos si se compara con los generados via ensamblado, sumado a menor complejidad y costo computacional son los indicadores que han sido considerados para tomar esta decision.

7.2 Matriz de confusion para el modelo escogido

7.3 Sensitividad, especificidad y precision

Observando la matriz de confusión, mas los estadísticos se pueden formular algunas observaciones:

- De 1986 registros totales, 1080 están siendo clasificados correctamente como no (es decir no se presencia dengue) y 776 como yes (se observa la presencia de dengue). 82 registros son falsos positivos y 48 falsos negativos.
- Accuracy obtenida es de 0.93 y considerando la proporción base, las variables estarían mejorando la medida de base.
- Sensitivity/recall: este indicador se observa en un valor de 0.92 lo cual indica que la probabilidad de detectar un valor positivo es alta. Un valor alto de recall indica que existirían menos falsos negativos.
- Specificity: La probabilidad de detectar correctamente los casos negativos es de un valor de 0.94.
- Precision: La probabilidad de acertar en caso de yes es de un 0.95

7.4 Tabla de parametros para regresion logistica

```
summary(logistica)
```

```
Call:
NULL
```

```
Deviance Residuals:
```

Min	1Q	Median	3Q	Max
-2.63747	-0.15358	-0.02882	0.44958	3.04419

```
Coefficients:
```

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	-1.8848	0.1866	-10.102	< 0.0000000000000002 ***
hiOpix	3.1524	0.2542	12.403	< 0.0000000000000002 ***
temp90	3.1225	0.7286	4.286	0.0000182 ***
Ymin	-1.8854	1.1527	-1.636	0.101914
Ymax	1.6698	1.1480	1.455	0.145781
temp	-2.6276	0.6948	-3.782	0.000156 ***
humid	3.7731	0.9470	3.984	0.0000677 ***
humid90	-3.4328	0.9786	-3.508	0.000452 ***
trees	-0.6970	0.1991	-3.500	0.000465 ***
trees90	0.5499	0.2221	2.477	0.013266 *

```
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
(Dispersion parameter for binomial family taken to be 1)
```

```
Null deviance: 2695.4  on 1985  degrees of freedom
Residual deviance: 1033.8  on 1976  degrees of freedom
AIC: 1053.8
```

```
Number of Fisher Scoring iterations: 7
```

- Observando la tabla de parámetros Ymin e Ymax no son variables relevantes para el modelo a pesar de que fueron seleccionadas previamente siguiendo los pasos de selección de variables con stepwise.

7.5 Puntos de corte

Se prueba cambiando punto de corte de 0.5 a 0.3

```
corte <- 0.3
salida_rf$predcorte <- ifelse(salida_rf$Yes>corte, "Yes", "No")
salida_rf$predcorte <- as.factor(salida_rf$predcorte)
confusionMatrix(salida_rf$obs, salida_rf$predcorte, positive = "Yes")
```

```
Confusion Matrix and Statistics
```

Prediction	Reference	
	No	Yes
No	1023	139
Yes	25	799

```
Accuracy : 0.9174
```

```

          95% CI : (0.9044, 0.9292)
No Information Rate : 0.5277
P-Value [Acc > NIR] : < 0.00000000000000022

      Kappa : 0.8333

McNemar's Test P-Value : < 0.00000000000000022

      Sensitivity : 0.8518
      Specificity : 0.9761
      Pos Pred Value : 0.9697
      Neg Pred Value : 0.8804
      Prevalence : 0.4723
      Detection Rate : 0.4023
      Detection Prevalence : 0.4149
      Balanced Accuracy : 0.9140

'Positive' Class : Yes

```

- Solo como experimento adicional, no se observa hay mejora, al contrario, la accuracy desciende de 0.93 a 0.91, pero si ha aumentado la especificidad, se sigue detectando bien los valores negativos.

7.6 Contraste de hipotesis entre algunos modelos

```

lista_medias <- rbind(medias1,medias2,medias3, medias4, medias5,medias6, medias11,  medias12,medias13, medias14,medias15, medias16)
modelos <- c("xgbm", "randomforest")
contraste <- lista_medias[which(lista_medias$modelo%in%modelos),]
resultados <- t.test(contraste$auc ~contraste$modelo)
resultados

```

```

Welch Two Sample t-test

data:  contraste$auc by contraste$modelo
t = -15.115, df = 6.6841, p-value = 0.000002043
alternative hypothesis: true difference in means between group randomforest and group xgbm is not equal to 0
95 percent confidence interval:
 -0.013022752 -0.009469958
sample estimates:
mean in group randomforest      mean in group xgbm
      0.9776175                0.9888639

```

7.7 Visualpred

Utilizando el script facilitado en los apuntes se realizan algunas comparaciones básicas con el paquete visualpred.

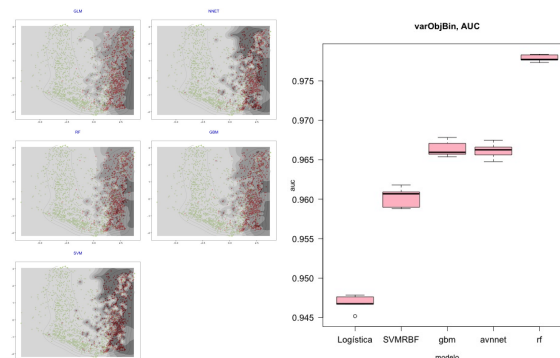


Figura 1: caption

- En general las clases se separan adecuadamente pero se observan algunas islas de observaciones sin clasificar correctamente. Con un modelado adicional en el script evaluando diferentes modelos y con criterio auc, random forest sigue mostrando buenos resultados y menor varianza que el resto de los modelos. ## Tabla resumen Como punto final a esta practica se adjunta tabla resumen con auc y tasa de fallos para cada modelo evaluado

```

tabla_final <- lista_medias %>%
  group_by(modelo) %>%
  summarise(tasa=mean(tasa))
knitr::kable(tabla_final, "pipe", caption = "Tasa de fallos ")

```

Cuadro 17: Tasa de fallos

modelo	tasa
Avnnet1	0.0734139
Avnnet2	0.1138973
Logistica1	0.1081571
Logistica2	0.1121853
Red con maxit	0.0724068
bagging	0.0726083
gbm	0.0862034
randomforest	0.0645519
svmLineal	0.1072508
svmPoly	0.0944612
svmRBF	0.0855992
xgbm	0.0494461

```
tabla_final <- lista_medias %>%
  group_by(modelo) %>%
  summarise(auc=mean(auc))
knitr::kable(tabla_final, "pipe", caption = "Accuracy ")
```

Cuadro 18: Accuracy

modelo	auc
Avnnet1	0.9717350
Avnnet2	0.9444342
Logistica1	0.9468125
Logistica2	0.9445338
Red con maxit	0.9722424
bagging	0.9753555
gbm	0.9691624
randomforest	0.9776175
svmLineal	0.9469829
svmPoly	0.9573743
svmRBF	0.9622408
xgbm	0.9888639

A Anexos

A.1 Otros metodos de seleccion de variables

Como un complemento a la selección de variables de tipo **Stepwise** que se ha estudiado en el apartado de selección de variables se ha decidido estudiar otras alternativas de selección de features utilizando diversas técnicas que se detallan a continuación:

A.1.1 Utilizando la libreria party y random forest

Un método de selección de variables es utilizando el algoritmo de random forest para encontrar un set de predictores, para eso se utiliza la librería *party* y se obtienen los siguientes resultados:

```
library(party)
s1 <- cforest(varObjBin ~ . ,
  data = dengue2,
  control = cforest_unbiased(mtry = 2, ntree = 50))
varimp(s1)
```

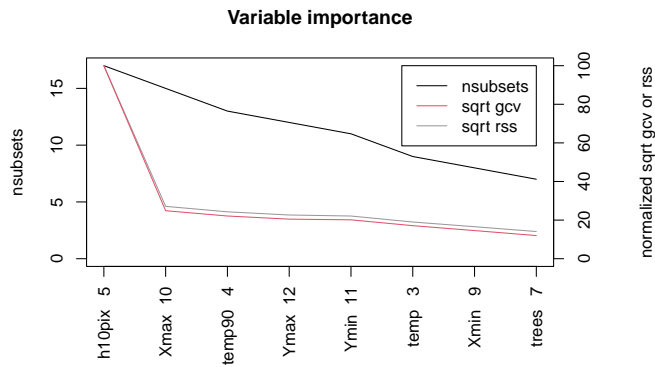
```
      humid      humid90      temp      temp90      h10pix      h10pix90
0.019011819 0.024480176 0.007135328 0.010393915 0.044306715 0.036816130
      trees      trees90      Xmin      Xmax      Ymin      Ymax
0.004198915 0.005095990 0.011027785 0.009075744 0.028311803 0.024437635
```

Agregar un comentario

A.1.2 Utilizando *MARS: Multivariate Adaptive Regression Splines*

Otro método de selección de variables es utilizando MARS, contenida en la librería *earth* para R

```
library(earth)
s2 <- earth(varObjBin ~ . , data=dengue2) # build model
evaluacion <- evimp(s2)
plot(evaluacion)
```

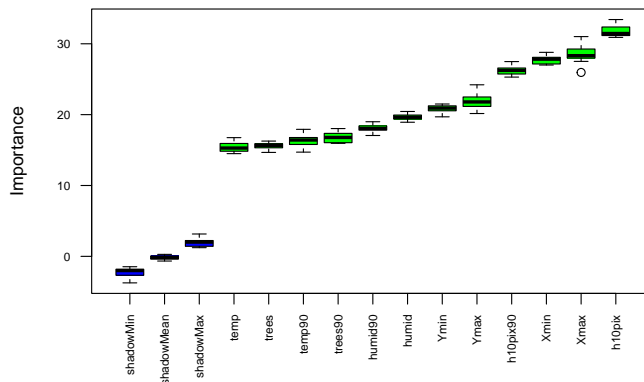


A.1.3 Utilizando la librería Boruta

Boruta es otra librería para realizar *feature selection* de forma automática, permitiendo un acercamiento inicial rápido al dataset (con sus ventajas y desventajas):

```
library(Boruta)
boruta_model <- Boruta(varObjBin ~ ., data= dengue2, doTrace=2)
relevancia <- names(boruta_model$finalDecision[boruta_model$finalDecision %in% c("Confirmed")])
print(relevancia)
```

```
[1] "humid" "humid90" "temp" "temp90" "h10pix" "h10pix90"
[7] "trees" "trees90" "Xmin" "Xmax" "Ymin" "Ymax"
plot(boruta_model, las = 2, cex.axis=0.7, xlab="")
```



Boruta permite revisar si el set de variables a estudiar entran en el modelo o podrían entrar de forma tentativa o definitivamente no han sido considerados como variables predictoras relevantes con lo cual también se pueden probar otros modelos y combinaciones (ej confirmados mas algún tentativo etc)

Para este dataset en particular la selección no marca variables como tentativas o descartadas por lo cual correspondería observar el gráfico generado y realizar algún corte de tipo manual en las variables.

A.1.4 Utilizando RFE con caret

Eliminación recursiva de características o RFE en sus siglas en ingles, es un estimador que asigna pesos a las características del dataset que se quiere estudiar

```
set.seed(7)
# Cargar datos
data(dengue2)
# Se define la funcion de control
controlrfe <- rfeControl(functions=rfunc, method="cv", number=4)
# Algoritmo RFE
res_rfe <- rfe(dengue2[,2:12], dengue2[,1], sizes=c(2:12), rfeControl=controlrfe)
#Resultados
print(res_rfe)
```

Recursive feature selection

Outer resampling method: Cross-Validated (4 fold)

Resampling performance over subset size:

Variables	RMSE	Rquared	MAE	RMSESD	RquaredSD	MAESD	Selected
2	0.2559	0.7144	0.13033	0.080328	0.188079	0.080015	
3	0.2103	0.8186	0.09381	0.008193	0.013435	0.001988	
4	0.2093	0.8200	0.09198	0.006153	0.011944	0.003804	
5	0.1927	0.8479	0.08635	0.009504	0.014010	0.004464	
6	0.1915	0.8499	0.08543	0.008331	0.011868	0.003713	*
7	0.1934	0.8473	0.08939	0.006407	0.009145	0.004466	
8	0.1941	0.8465	0.09210	0.004874	0.007074	0.003773	
9	0.1949	0.8450	0.09183	0.004130	0.007247	0.003856	
10	0.1985	0.8395	0.09571	0.004161	0.006483	0.003471	
11	0.2012	0.8353	0.09888	0.002943	0.007213	0.002814	

The top 5 variables (out of 6):

Xmax, h10pix, Xmin, h10pix90, Ymin

```
# Features escogidas
```

```
predictors(res_rfe)
```

```
[1] "Xmax"      "h10pix"    "Xmin"      "h10pix90" "Ymin"      "temp"
```

```
# graficos
```

```
# plot(results, type=c("g", "o"))
```

Con las variables que ha seleccionado el modelo de RFE mas los modelos anteriores se prueban con la función `cruzadalogistica()` y se genera el boxplot para comparar:

```
# Generado con libreria party
anexo1 <- cruzadalogistica(
  data = data,
  vardep = "varObjBin", listconti = c("humid", "humid90","temp", "temp90",
    "h10pix", "h10pix90"),
  listclass = c(""),
  grupos = 4, inicio = 1234, repe = 5)

anexo1$modelo <- "log-party"

# Generado con MARS
anexo2 <- cruzadalogistica(
  data = data,
  vardep = "varObjBin",
  listconti = c("h10pix", "Xmax", "temp90", "Ymax", "Ymin", "temp", "Xmin",
    "trees"),
  listclass = c(""),
  grupos = 4, inicio = 1234, repe = 5)

anexo2$modelo <- "log-mars"

# Generado con Boruta
anexo3 <- cruzadalogistica(
  data = data,
  vardep = "varObjBin",
  listconti = c("h10pix", "Xmax", "Xmin", "h10pix90", "Ymax"),
  listclass = c(""),
  grupos = 4, inicio = 1234, repe = 5)

anexo3$modelo <- "log-boruta"

# Generado con RFE
anexo4 <- cruzadalogistica(
  data = data,
  vardep = "varObjBin",
  listconti = c("Xmax", "h10pix","Xmin","h10pix90","Ymin","temp" ),
  listclass = c(""),
  grupos = 4, inicio = 1234, repe = 5)

anexo4$modelo <- "log-RFE"
```

Conclusiones:

- A modo general la selección de variables realizada con `stepwiseAIC` (Logistica1), sigue siendo la que presenta mejor AUC y menor tasa de fallos, ademas de menor varianza por lo que se seguirá trabajando con ese modelo.

A.2 Árboles de clasificacion

A.2.1 Selecccion de variables

Como complemento al trabajo realizado anteriormente con los algoritmos de ML y su posterior evaluación, se ha añadido un apartado de practica modelando arboles de clasificación utilizando R y el dataset *dengue*

Primer modelado, modelo con todas las variables del dataset para realizar estudio de importancia de variables (que se hizo en el apartado de selección con Stepwise pero se prefiere agregarlo a la practica)

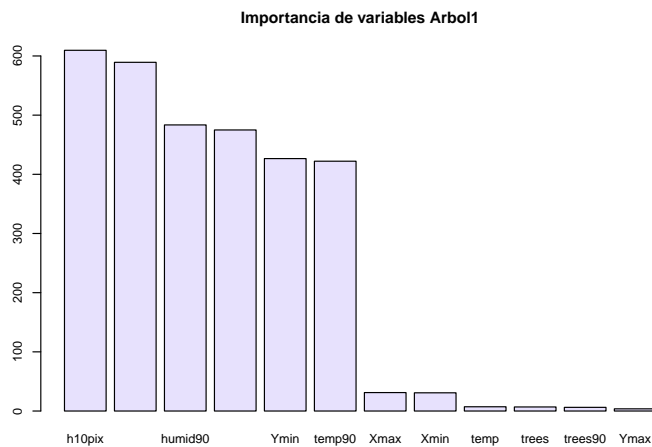
Preguntar si en arboles trabajo con las variables que da el modelo o me ajusto a lo que dio la selección de variables con stepwise

```
# Modelo completo con gini para variable dependiente categorica
arbol1 <- rpart(factor(varObjBin) ~ .,
  data = dengue_arbol,
  minbucket = 30,
  method = "class",
  parms = list(split = "gini"))

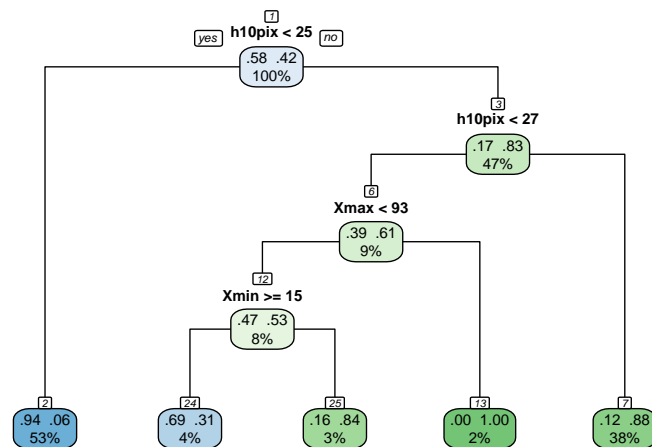
# Reglas de decision
rattle::asRules(arbol1)
```

Representación gráfica de las variables consideradas relevantes y árbol:

```
# Importancia de variables
par(cex = 0.7)
barplot(height = arbol1$variable.importance, col = "#e1f5fe",
  main = "Importancia de variables Arbol1")
```



```
# Ploteo del arbol
rpart.plot(arbol1, extra = 105, tweak = 1.2, type = 1, nn = TRUE)
```



Observaciones:

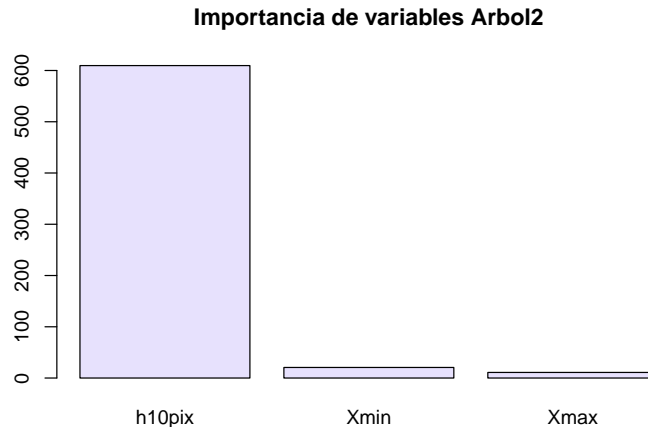
- Según el gráfico importancia de variables Árbol1, las variables predictoras mas relevantes serian, h10pix, h10pix90, humid90, Ymin, humid e Ymax.
- agregar observaciones del árbol

Se prueba modelando un nuevo árbol esta vez usando maxsurrogate = 0, solo como complemento puesto que el archivo dengue no contiene datos faltantes:

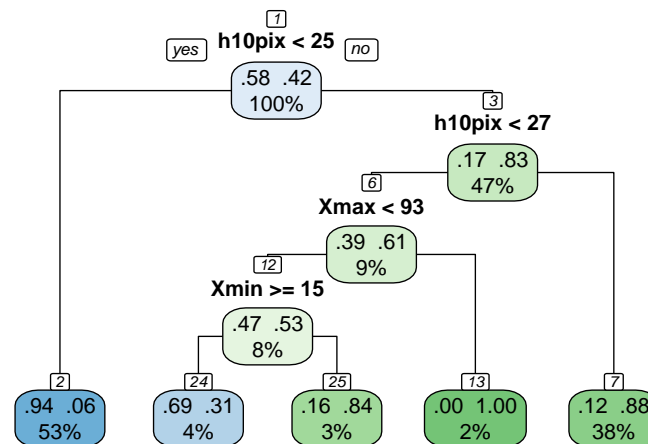
```
# Modelo con maxsurrogate = 0 para que trabaje los na's
arbol2 <- rpart(factor(varObjBin) ~ .,
  data = dengue_arbol,
  minbucket = 30,
  method = "class",
  maxsurrogate = 0,
  parms = list(split = "gini"))
```

Representación gráfica de las variables consideradas relevantes y árbol:

```
# Importancia de variables
barplot(height=arbol2$variable.importance, col = "#e1efd",
  main = "Importancia de variables Arbol2")
```



```
# Ploteo del arbol
rpart.plot(arbol2, extra = 105, tweak = 1.2, type = 1, nn = TRUE)
```



Observaciones:

- Según el gráfico importancia de variables Arbol2, las variables predictoras mas relevantes serian, h10pix e Ymax.
- agregar observaciones del árbol

A.2.2 Tuning de algoritmos usando rpart()

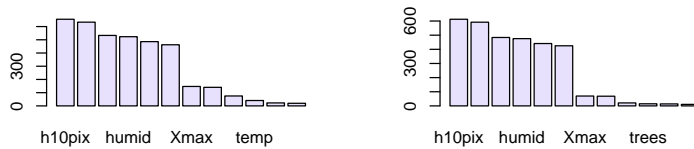
De forma inicial se realiza un primer tuning de arboles observando el comporta miento de la complejidad del árbol graficando su estructura en relación al parámetro minbucket (5, 30 y 60) utilizando la función rpart()

```
# Tuneado sobre complejidad del arbol con Rpart
arbol_min5 <- rpart(factor(varObjBin) ~ ., data = dengue_arbol, minbucket = 5, cp = 0)
arbol_min30 <- rpart(factor(varObjBin) ~ ., data = dengue_arbol, minbucket = 30, cp = 0)
arbol_min60 <- rpart(factor(varObjBin) ~ ., data = dengue_arbol, minbucket = 60, cp = 0)
```

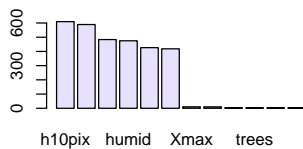
Representación gráfica del tuning de minbucket

```
par(mfrow= c(2,2))
barplot(height = arbol_min5$variable.importance, col = "#e7e1fd",
        main = "Importancia de variables con minbucket 5")
barplot(height = arbol_min30$variable.importance, col = "#e7e1fd",
        main = "Importancia de variables con minbucket 30")
barplot(height = arbol_min60$variable.importance, col = "#e7e1fd",
        main = "Importancia de variables con minbucket 60")
par(mfrow= c(2,2))
```

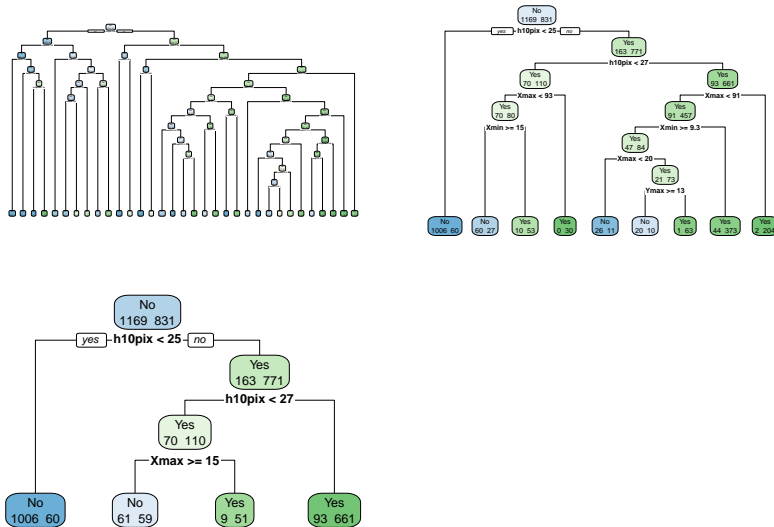
Importancia de variables con minbucket Importancia de variables con minbucket



Importancia de variables con minbucket



```
rpart.plot(arbol_min5, extra = 1)
rpart.plot(arbol_min30, extra = 1)
rpart.plot(arbol_min60, extra = 1)
```



Observaciones

Se puede apreciar que con minbucket 5 se genera un modelo complejo y con minbucket 60 un modelo básico, para encontrar un equilibrio y ver otras opciones de minbucket a continuación se profundiza en el tuneado utilizando caret de R

A.2.3 Tuning de algoritmos usando caret()

Como indican los apuntes, no se puede realizar tuning en el grid por lo que se construye un bucle for() que recorre por cada numero de minbucket indicado y va guardando los resultados para una posterior evaluación:

```
# En los ejemplos anteriores se dejaron los na's para ver el comportamiento, se eliminan
# tuning de minbucket en bucle
# Tratamiento de datos faltantes
```

```
dengue_arbol1 <- na.omit(dengue_arbol, (!is.na(dengue_arbol)))
colSums(is.na(dengue_arbol1)) > 0

# Tuning con minbucket en bucle
set.seed(12345)
control_arbol <- trainControl(method = "repeatedcv", number = 4,
  repeats = 5, savePredictions = "all", classProbs = TRUE)

arbolgrid <- expand.grid(cp = c(0))
tabla_resultados <- c()

for (minbu in seq(from = 5, to = 60, by = 5)){
  arbolgrid <- expand.grid(cp=c(0))
  arbolcaret <- train(factor(varObjBin) ~ h10pix + temp90 + Ymin + Ymax +
    temp + humid + humid90 + trees + trees90,
    data = dengue_arbol1,
    method = "rpart",
    minbucket = minbu,
    trControl = control_arbol,
    tuneGrid = arbolgrid)

  accuracy <- arbolcaret$results$Accuracy
  sal <- arbolcaret$pred
  salconfu <- confusionMatrix(sal$pred, sal$obs)
  curvaroc <- roc(response = sal$obs, predictor = sal$Yes)
  auc <- curvaroc$auc
  # Guarda los resultados en formato tabla
  tabla_resultados <- rbind(tabla_resultados, c(minbu, accuracy, auc))
}

# Cambia los nombres de las columnas
colnames(tabla_resultados) <- c("minibucket", "accuracy", "AUC")
```

Con el código anterior se genera una tabla para una mejor visualización de los indicadores:

```
knitr::kable(tabla_resultados, "pipe")
```

minibucket	accuracy	AUC
5	0.9061411	0.9570891
10	0.9012119	0.9510540
15	0.9040294	0.9534192
20	0.9041276	0.9542344
25	0.9036242	0.9565679
30	0.9087667	0.9553399
35	0.9048338	0.9535379
40	0.9039296	0.9537842
45	0.9017137	0.9508535
50	0.9071493	0.9538240
55	0.9012103	0.9539353
60	0.9042310	0.9557522

Como se puede apreciar en la tabla anterior, en combinación entre mejor accuracy y AUC para este modelo es adecuado utilizar un minbucket de 30. Con ese numero se realiza la validación cruzada repetida utilizando la función `cruzaaarbolbin()`

```
set.seed(12345)
control_arbol1 <- trainControl(method = "cv", number = 4,
  savePredictions = "all", classProbs = TRUE)

arbol4 <- train(factor(varObjBin) ~ h10pix + temp90 + Ymin + Ymax + temp +
  humid + humid90 + trees + trees90,
  data = dengue_arbol1,
  method = "rpart",
  minbucket = 20,
  trControl = control_arbol1,
  tuneGrid = arbolgrid)
```

Se obtiene la matriz de confusión para este modelo

```
salida_a4 <- arbol4$pred
salida4_confusion <- confusionMatrix(salida_a4$pred, salida_a4$obs)
salida4_confusion
```

Confusion Matrix and Statistics

```

      Reference
Prediction  No  Yes
No      1060   76
Yes      102  748
```

```

Accuracy : 0.9104
95% CI : (0.8969, 0.9226)
```

```

No Information Rate : 0.5851
P-Value [Acc > NIR] : < 0.0000000000000002

Kappa : 0.8162

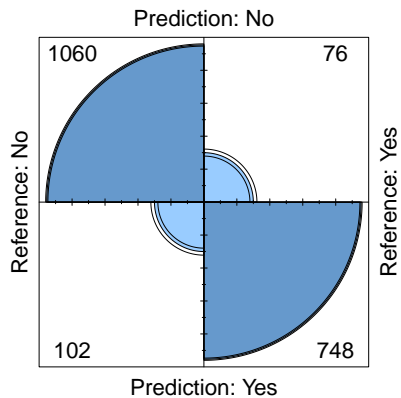
McNemar's Test P-Value : 0.06095

Sensitivity : 0.9122
Specificity : 0.9078
Pos Pred Value : 0.9331
Neg Pred Value : 0.8800
Prevalence : 0.5851
Detection Rate : 0.5337
Detection Prevalence : 0.5720
Balanced Accuracy : 0.9100

'Positive' Class : No

fourfoldplot(salida4_confusion$table)

```

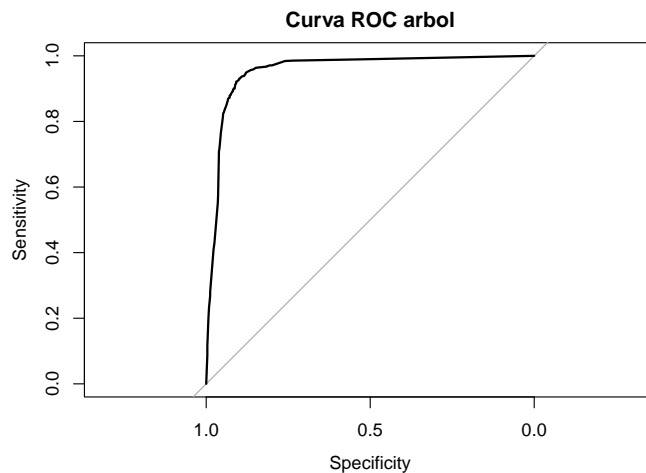


Se obtiene la curva ROC

```

# Curva roc
curvaroc <- roc(response = salida_a4$obs, predictor = salida_a4$Yes)
auc <- curvaroc$auc
plot.roc(roc(response = salida_a4$obs, predictor = salida_a4$Yes),
        main = "Curva ROC arbol")

```



A.2.4 Comparacion con otros algoritmos de ML

Con los resultados obtenidos de entrenar el modelo se realiza la validación cruzada repetida para posteriormente comparar las medias con los demás modelos y evaluar sesgo-varianza

```

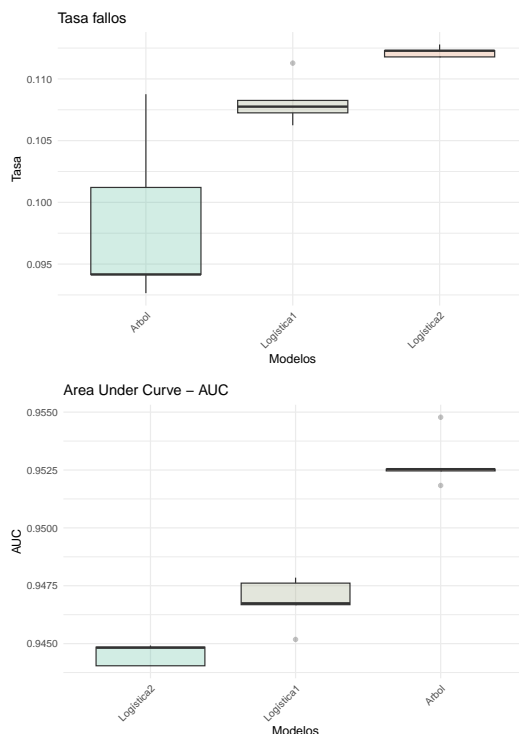
# 2. Validacion cruzada repetida -----
# Con los resultados de la tabla con iteraciones se configura la validacion cruzada
medias_arbol <- cruzadaarbolbin(
  data = dengue_arbol1,
  vardep = "varObjBin",
  listconti = c("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax", "temp",
    "humid", "humid90"),
  listclass = c(""), grupos = 4, inicio = 1234, repe = 5,
  cp = c(0), minbucket = 20)

```

```
medias_arbol$modelo <- "Arbol"
```

Con la función `genera_gráficos()` se despliega la comparación de medias para los distintos algoritmos entrenados

```
#genera_graficos(medias1, medias2, medias3, medias4, medias5,medias6, medias9,medias11, medias_arbol)
genera_graficos(medias1, medias2, medias_arbol)
```



Observaciones:

- Para esta practica en particular, un árbol no es lo suficientemente competitivo frente a otros algoritmos modelados. Presenta mejores indicadores que la regresión logística o una red neuronal pero ante otros algoritmos como bagging o random forest se queda un tanto atrás.

A.3 Librería h2o para python

Como practica adicional se ha decidido realizar un modelado con autoML utilizando la librería h2o para python, utilizando un jupyter notebook para este fin.

El notebook se puede revisar en detalle en: https://github.com/TiaIvonne/Machine-LearningR/blob/master/h2o%20python/autoML_dengue.ipynb

El primer paso ha sido generar un modelo de autoML que determine cual combinación es la mas adecuada para el set de datos dengue y buscar si hay coincidencias con la practica realizada con R.

Para autoML la mejor combinación es utilizar un Xgboost el cual coincide con los resultados de esta practica (a pesar de que se dio por ganador a un random forest) con metricas de buena calidad y similares a las obtenidas con R.