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

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. Dimension de los datos: 2000 filas x 14 columnas

2 Analisis exploratorio de los datos

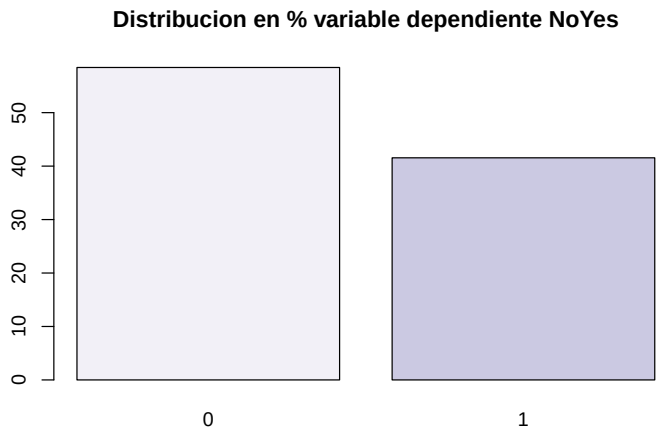
X	humid	humid90	temp	temp90	h10pix	h10pix90	trees	trees90	NoYes	Xmin	Xmax	Ymin	Ymax
1	0.6713889	4.416667	2.037500	8.470835	17.35653	17.80861	0	1.5	0	70.5	74.5	38.0	35.5
2	7.6483340	8.167499	12.325000	14.925000	10.98361	11.69167	0	1.0	0	62.5	64.5	35.5	34.5
3	6.9790556	9.563057	6.925000	14.591660	17.50833	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
```

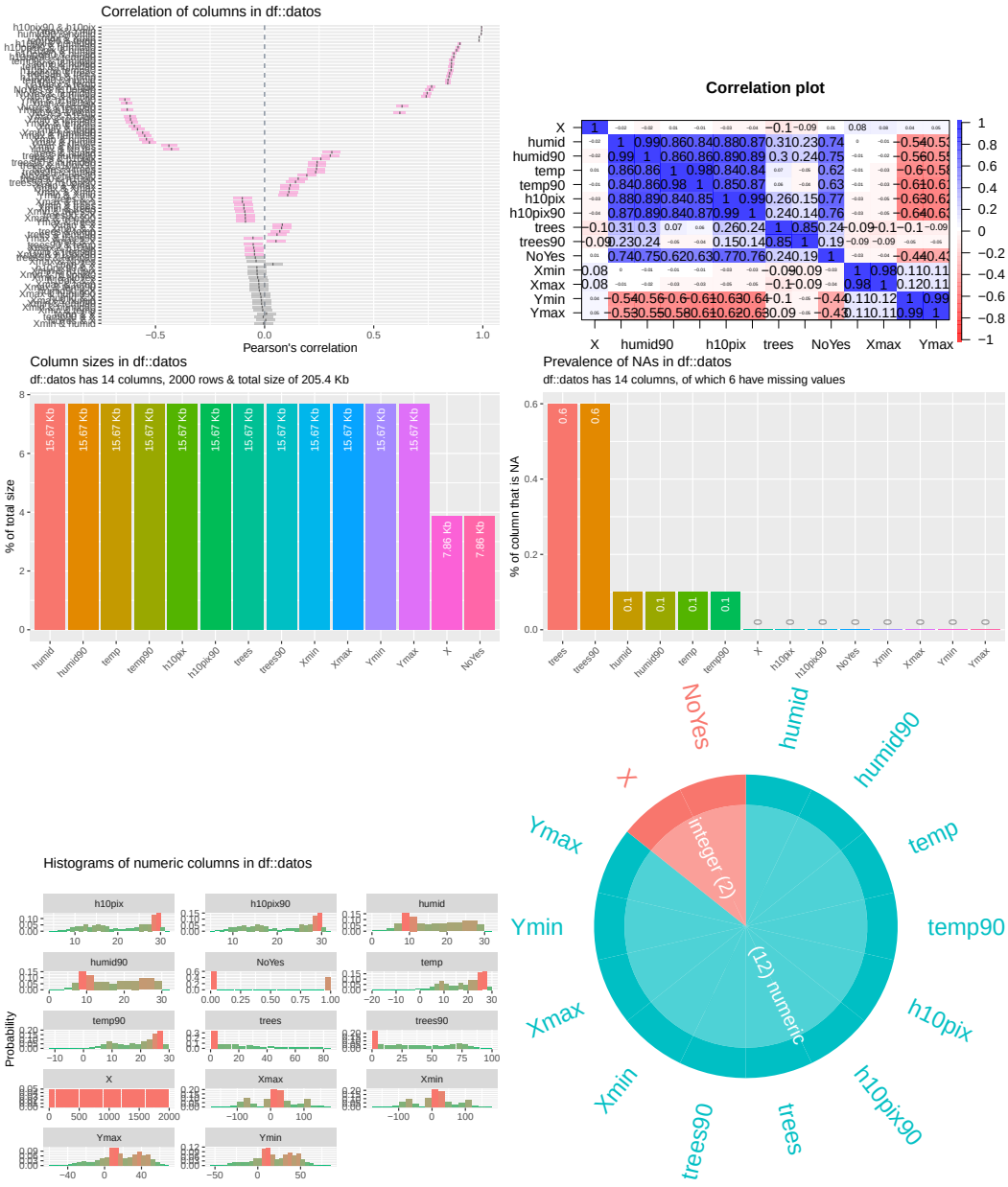
Como se observa anteriormente, son 14 columnas con 2000 filas, los tipos de datos son de tipo numérico incluyendo la variable objetivo **NoYes,** se realiza a continuacion una exploración mas detallada del dataset a estudiar.

En cuanto a las observaciones sobre la variable independiente (*NoYes*) y observando el grafico de referencia, se observa un dataset un tanto desbalanceado pero no deberia afectar en demasia la ejecucion de esta practica.

NoYes	cuenta	porcentaje
0	1169	58.45
1	831	41.55



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



Observaciones:

- Correlacion:
- Tipos de datos por columna: Datos tipo numeric e integer incluyendo la variable estudio.
- NA's: No se presenta un % alto de datos ausentes, en la siguiente seccion se trabajara en ello. trees y trees90 presentan un NA cercano al 0.6%
- Histograma:

3 Transformaciones

Se realizan transformaciones sobre los datos, se eliminan columnas, tratamiento de datos faltantes y discretización de los datos:

```
# 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, dummys etc)
base <- data.frame(varObjBin, temp)

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

varObjBin	humid	humid90	temp	temp90	h10pix	h10pix90	trees	trees90	Xmin	Xmax	Ymin	Ymax
No	-2.187817	-1.752592	-2.04603997	-1.42684042	-0.5252414	-0.5182986	-0.9516725	-1.173965	0.9177821	0.9477775	0.7759536	0.7328488
No	-1.235570	-1.240199	-0.76267048	-0.58677814	-1.3943124	-1.3621731	-0.9516725	-1.191357	0.7886923	0.7859356	0.6691044	0.6904187
No	-1.326917	-1.049555	-1.43632247	-0.63016510	-0.5045405	-0.5435904	-0.9516725	-1.184400	0.8855096	0.8668566	0.6904742	0.7116337
No	-2.127896	-2.106585	-1.72116940	-1.74236542	-0.5169085	-0.5585368	-0.9516725	-1.205271	0.8613053	0.8425803	0.6477345	0.6692036
No	-1.047396	-1.025007	-0.03288072	0.03602869	-1.0043711	-1.0653740	-0.9516725	-1.226141	0.7644880	0.7859356	0.5836250	0.5843434
No	-1.062811	-1.055877	-0.81568941	-0.36312211	-1.2977547	-1.3621731	-0.9516725	-1.219184	0.8209648	0.8021198	0.7118441	0.7116337

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

4 Seleccion de variables

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

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

# Seleccion 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 mas 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:

```
# Seleccion AIC
listaAIC <- steprepetidobinaria(
    data = data,
    vardep = vardep,
    listconti = listconti,
    sinicio = 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":

```
# Seleccion con Bic
listaBIC <- steprepetidobinaria(
    data = data,
    vardep = vardep,
    listconti = listconti,
    sinicio = 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]])
```

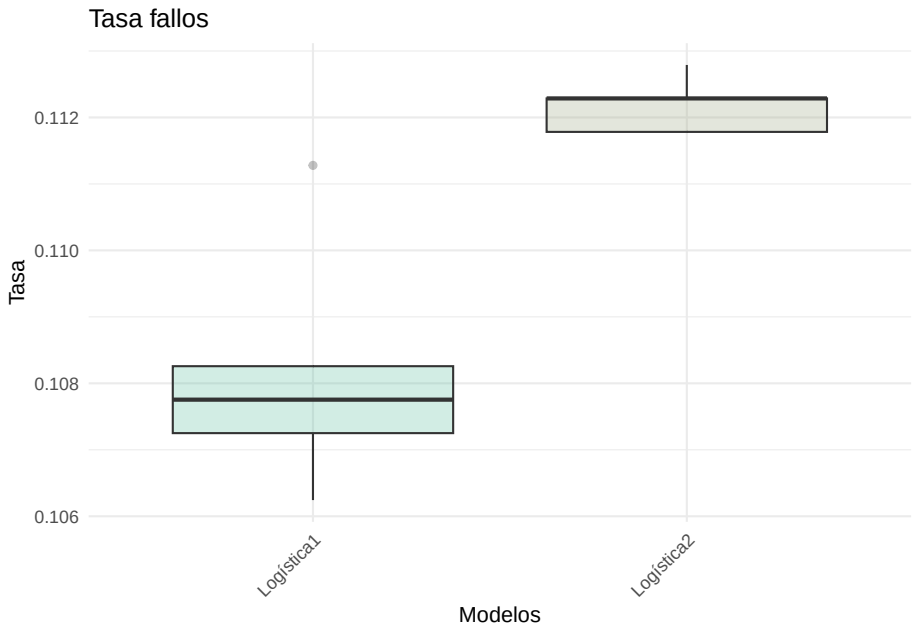
```
c("h10pix", "humid")
```

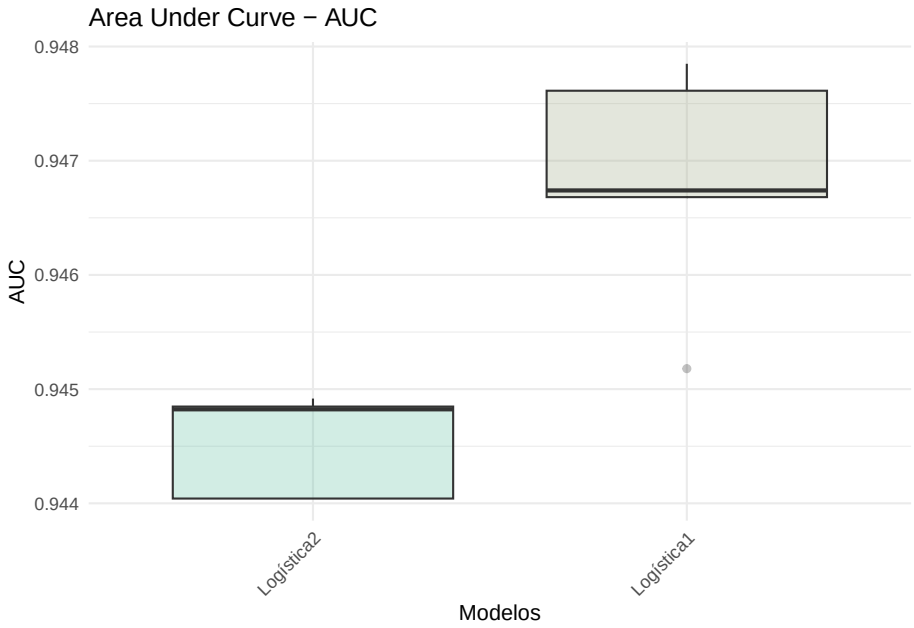
Observando los resultados anteriores y obteniendo los set de variables se comparan con validación cruzada repetida:

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

medias1$modelo <- "Logística1"

# Seleccion de variables BIC
medias2 <- cruzadalogistica(
    data = data,
    vardep="varObjBin",
    listconti = c("h10pix", "temp90"),
    listclass = c(""),
    grupos = 4,
    sinicio = 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 agrega como anexo):

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





Observaciones:

- El modelo de Logistica1 entrega mejores resultados que el modelo2

5 Algoritmos de ML para clasificación binaria

5.1 Redes neuronales

5.1.1 Tunning de redes neuronales

Una vez seleccionada las variables utilizando StepwiseAIC para logística, se comienza con el tunning 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 esta 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 Tunning 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")

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

# Con las variables seleccionadas stepwiseAIC
redavnnet1 <- train(
    varObjBin ~ h10pix + temp90 + trees + trees90 + Ymin + Ymax + temp +
    humid + humid90,
    data = dengue,
    method = "avNNet",
    linout = FALSE,
    maxit = 100,
    trControl = control_redes,
    tuneGrid = avnnetgrid,
    repeats = 5,
    trace = F)

knitr::kable(redavnnet1$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 o

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

```
# Con las variables seleccionadas BIC
redavnnet2 <- train(
    varObjBin ~ h10pix + temp90 ,
    data = data,
    method = "avNNet", linout = FALSE, maxit = 100,
    trControl = control_redes, tuneGrid = avnnetgrid,
    repeats = 5,
    trace = F)

knitr::kable(redavnnet2$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

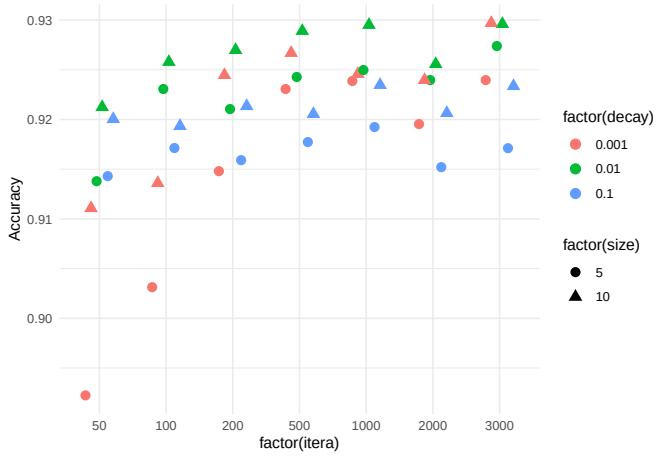
Agregar observaciones

5.1.3 Tunning de redes neuronales con MAXIT

Otro criterio para realizar tunning 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("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax",
    "temp", "humid", "humid90")

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

control_maxit <- trainControl(method = "cv", number = 4, savePredictions = "all")

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 tunning 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 tunning con criterio accuracy y controlando maxit = se generan las correspondientes validaciones cruzadas repetidas utilizando la función cruzadaavnnetbin()

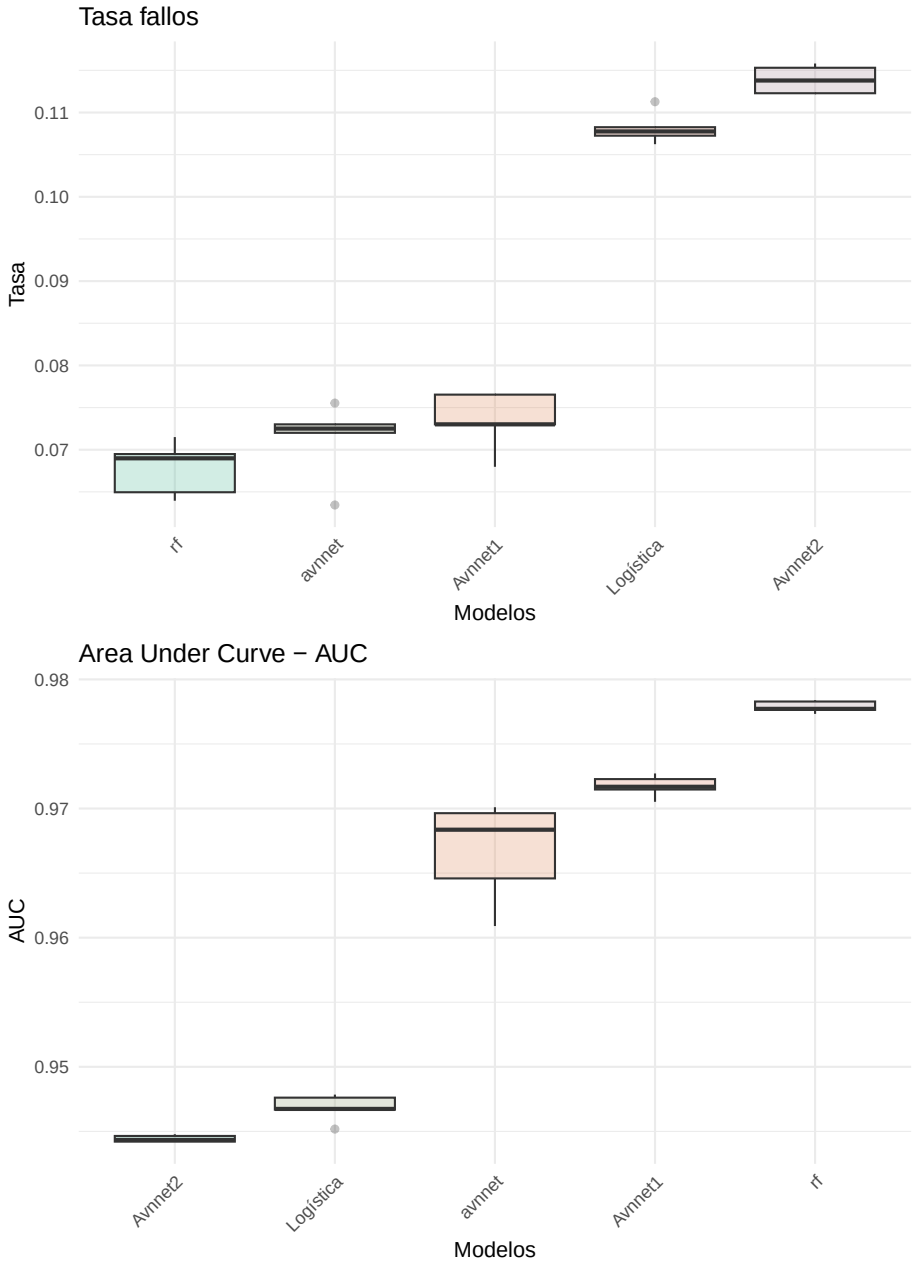
```
# Validacion cruzada repetida con criterio accuracy
medias3 <-cruzadaavnnetbin(
    data = dengue,
    vardep = "varObjBin",
    listconti = c("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax",
    "temp", "humid", "humid90"),
    listclass = c(""),
    grupos = 4,
    sinicio = 1234,
    repe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

medias3$modelo <- "Avnnet1"

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

medias4$modelo <- "Avnnet2"

# Validacion cruzada repetida tunning maxit
medias5 <- cruzadaavnnetbin(
    data = data2,
```

```
    vardep = "varObjBin",
    listconti = c("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax",
    "temp", "humid", "humid90"),
    listclass = c(""),
    grupos = 4,
    sinicio = 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 tunning con maxit.
- De todos modos esta es una comparativa inicial y aun se deben entrenar nuevos algoritmos y modelados para tomar una decisión.

5.2 Bagging

5.2.1 Tunning de bagging

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

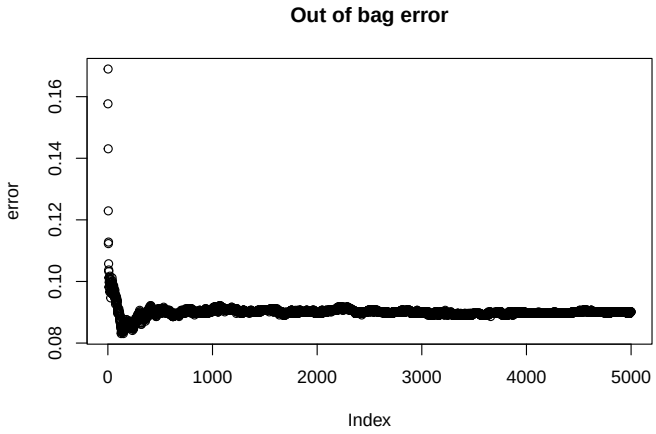
Primer modelo de bagging con variables previamente seleccionadas

```
rfgrid <- expand.grid(mtry = c(6))
set.seed(12345)
control_bagging <- trainControl(
    method = "cv",
    number = 4,
    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	0.9007989	0.7987109	0.008227	0.0164896

A continuación se explora el Out of bag error para este modelo, buscando parámetros para optimizacion, tal como lo indican los apuntes, se debe utilizar la función de randonForest() 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:

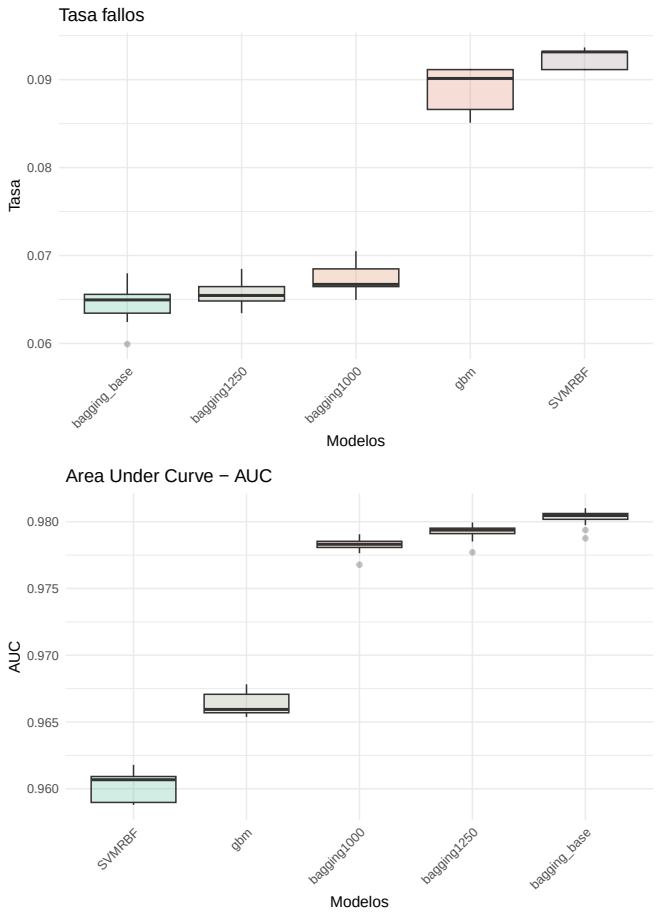
Nota: El bagging medias7 o bagging_base es solo como adicional pues se hace con 10 grupos de validación cruzada, es solo para observar el efecto y comparar. medias8, medias9 y medias10 si se consideran pues utilizan 4 grupos de validación y con diversos tamaños muestrales:

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

medias7$modelo <- "bagging_base"

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

```
medias8$modelo <- "bagging1000"

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

medias9$modelo <- "bagging1250"

# Bagging manipulando sampsize 1500
medias10 <-cruzadarfbin(
    data = dengue,
    vardep = "varObjBin",
    listconti = c("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax",
    "temp", "humid", "humid90"),
    listclass = c(""),
    grupos = 10,
    sinicio = 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 tamanio muestral de bagging
genera_graficos(medias6, medias7, medias8, medias9, medias10, medias10)
```



Observaciones

- De todo el grupo y descartando bagging_base que no se considera, los mejores resultados están entre bagging 1250 y bagging1500. Para comparar modelos en el apartado final incluyendo los modelos de random forest se deja el modelo medias9.

5.3 Random Forest Classifier

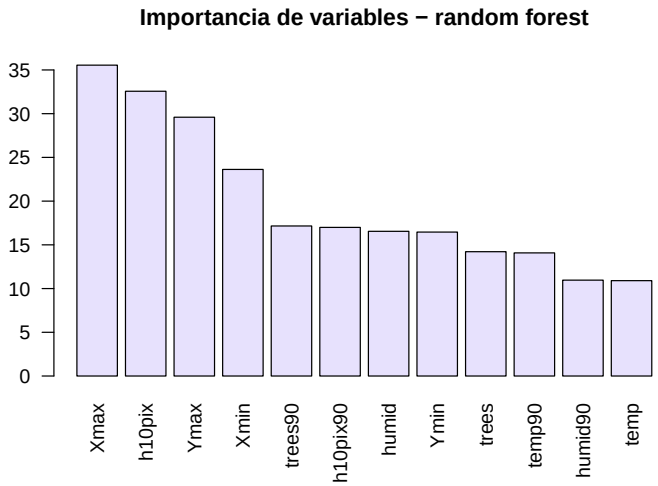
5.3.1 Tunning de random forest

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 = "cv",
    number = 4,
    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



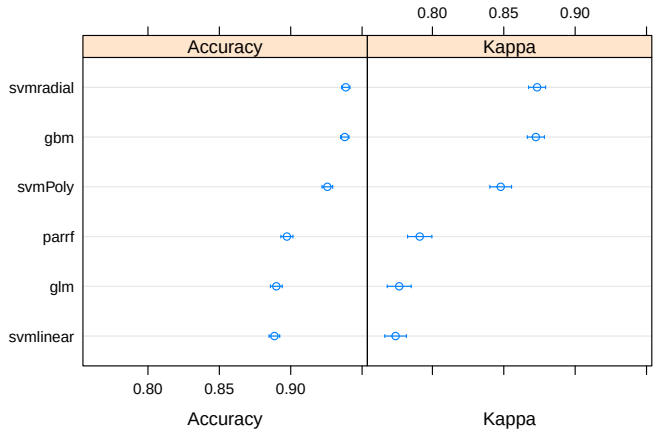
Para el modelo anterior el mejor mtry es 9 y en el gráfico se observa que las variables predictoras mas relevantes serian: Xmax, h10pix, Ymax, trees90, h10pix90, humid, Ymin lo cual difieren a los resultados entregados por la selección *stepwise* pero para explorar las alternativas que ofrece el algoritmo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



Confidence Level: 0.95

```
fit$bestTune
```

```
  mtry
2    4
```

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

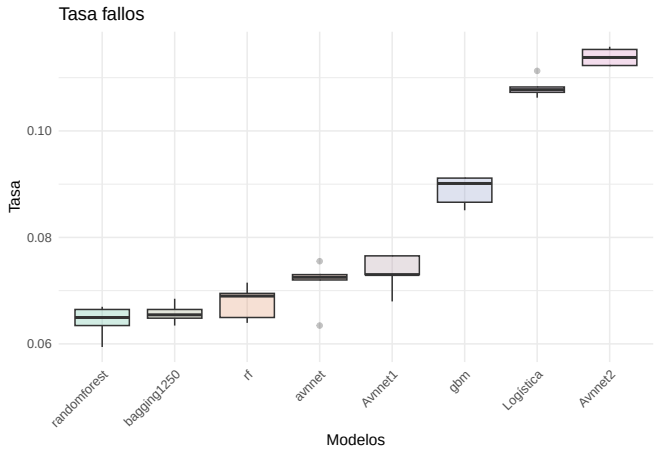
Se genera la validación cruzada para random forest utilizando la función cruzadarfbin y los parámetros obtenidos del estudio anterior, con mtry= 4 y ntree = 500

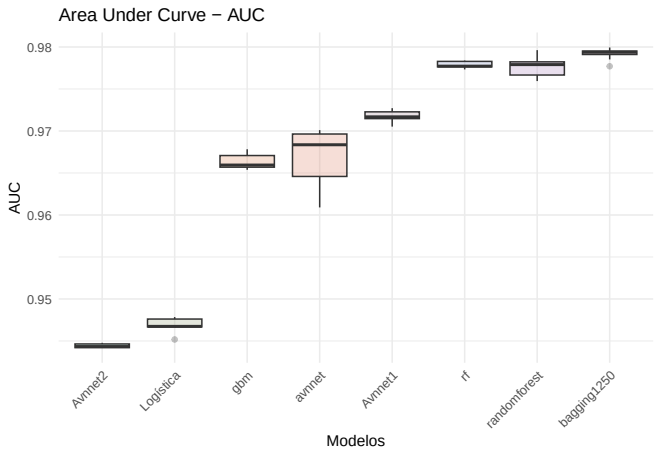
```
# Probar a dejar el parametro sampsize sin valor para que lo haga x defecto
medias11 <- cruzadarfbin(data = dengue, vardep = "varObjBin",
    listconti = c("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax",
    "temp", "humid", "humid90"),
    listclass = c(""),
    grupos = 4,
    sinicio = 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 el momento, randomforest (que muestra mas varianza comparado con los otros algoritmos) 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

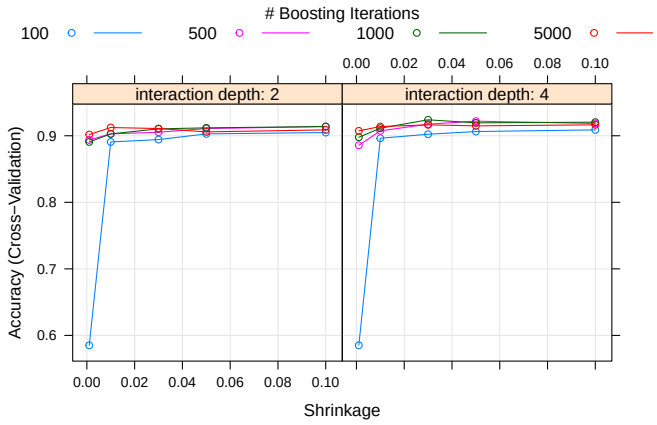
5.4.1 Tunning de gradient boosting

Se realiza un primer tunning del algoritmo gradient boosting con caret, probando los parametros basicos 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 = "cv",
    number = 4,
    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
```

```
   n.trees interaction.depth shrinkage n.minobsinnode
23    1000                 4      0.03             10
```



Observaciones:

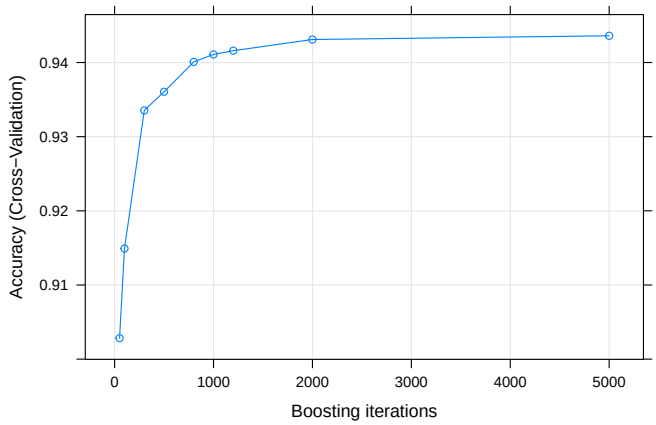
- Segun gbm$bestTune los mejores resultados se consiguen con 1000 iteraciones, shrinkage de 0.1 y n.minobsinnode de 5. Los graficos muestran que en iteraciones desde 500 en adelante no se observan diferencias considerables y tienen a compartir resultados especialmente desde el shrinkage 0.06. El maximo accuracy estaria 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

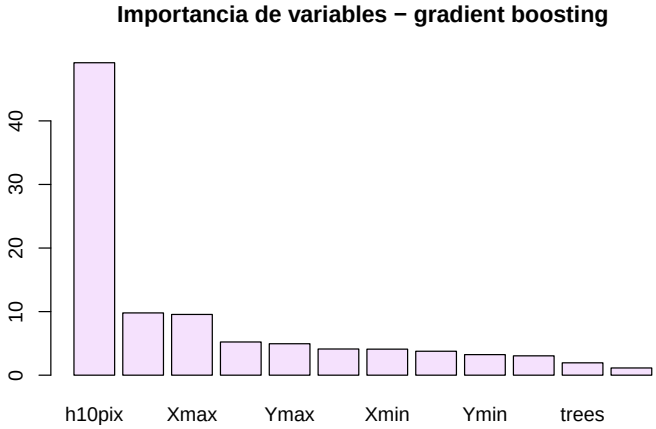
Se prueba con algunos parametros para buscar en que iteracion se estabiliza el algoritmo

```
# 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       0.1              5
```



Importancia de variables



Observaciones:

- Pendiente, preguntar si se usan todas las variables o las seleccionadas con stepwise

5.4.2 Validacion cruzada para gradient boosting

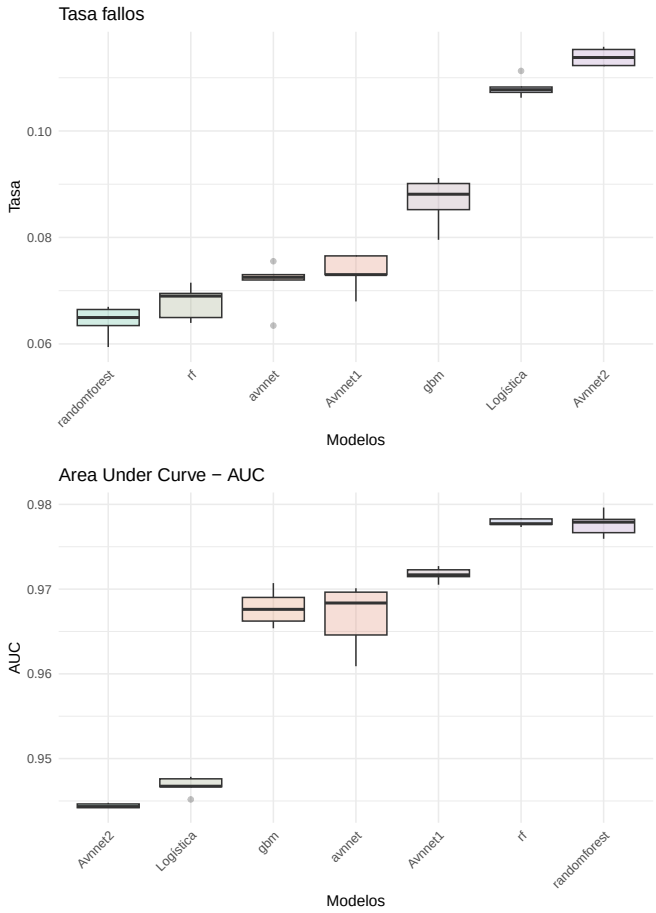
Se genera la validación cruzada para gradient boosting utilizando los parametros obtenidos en el proceso anterior de tunning:

```
# 2.Validacion cruzada repetida gradient ----------------------------------------
# Segun el grafico entre 2000 y 5000 no hay tanta diferencia, se deja en 3000
medias12 <- cruzadagbmbin(
    data = dengue,
    vardep="varObjBin",
    listconti=c("h10pix", "temp90", "trees", "trees90", "Ymin", "Ymax",
    "temp", "humid", "humid90"),
    listclass = c(""),
    grupos = 4,
    sinicio = 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_graficos(medias1, medias2, medias3, medias4, medias5,medias6, medias11,
                medias12)
```



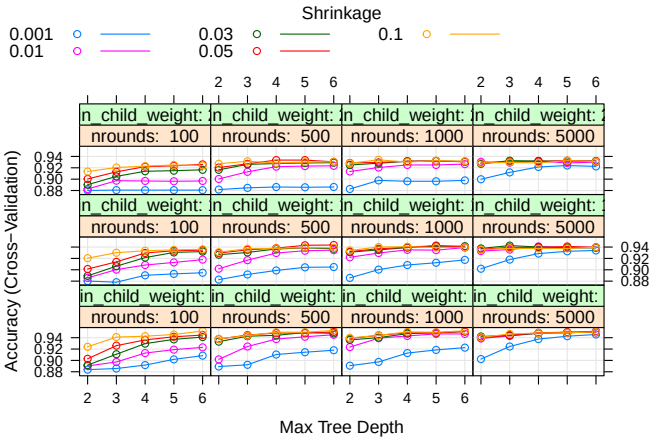
Observaciones:

- gbm() tiene un buen auc y esta en el medio del grafico de la tasa de fallos, de todas maneras no alcanza los niveles que muestra un random forest o un bagging para este dataset.

5.5 Xgboost

5.5.1 Tunning 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 = "cv",
    number = 4,
    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
```



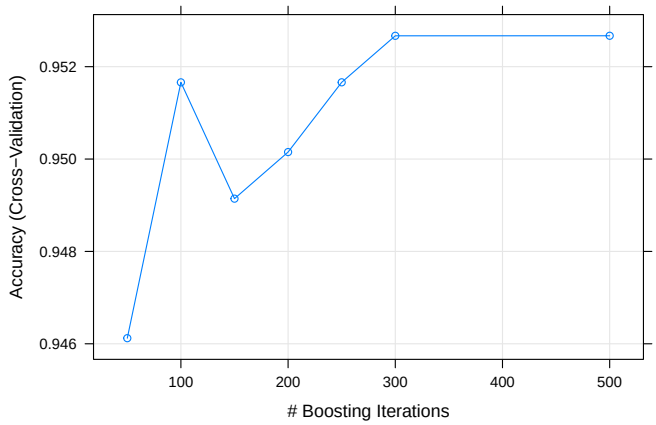
	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
290	500	6	0.1	0	1	5	1

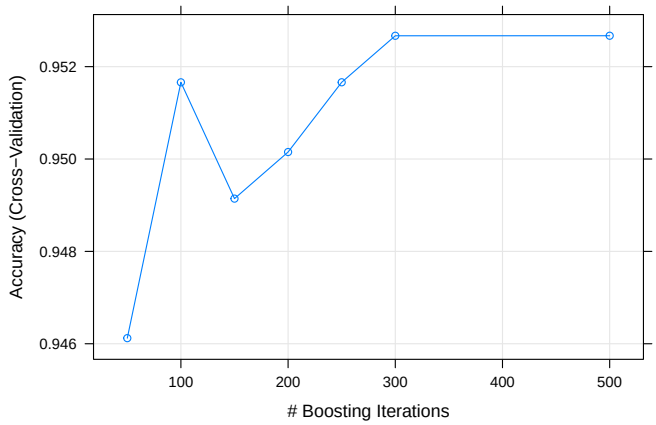
Observaciones:

- Pendiente, es necesario hacer feature selection?

```
# 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 = "cv",
    number = 4,
    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 = "cv",
    number = 4,
    savePredictions = "all",
    classProbs = TRUE)

xgbm3 <- train(
    factor(varObjBin) ~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
varImp(xgbm3)
plot(varImp(xgbm3))
```

5.5.1.1 Estudio de early stopping con xgboost





	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
7	500	6	0.1	0	1	5	1

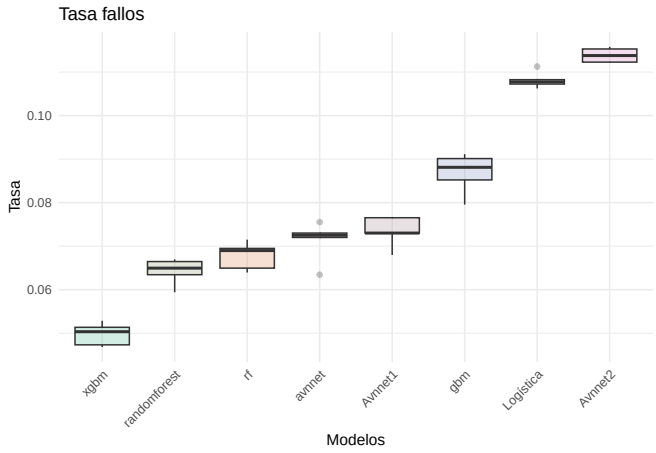
	nrounds	max_depth	eta	gamma	colsample_bytree	min_child_weight	subsample
7	500	6	0.1	0	1	5	1

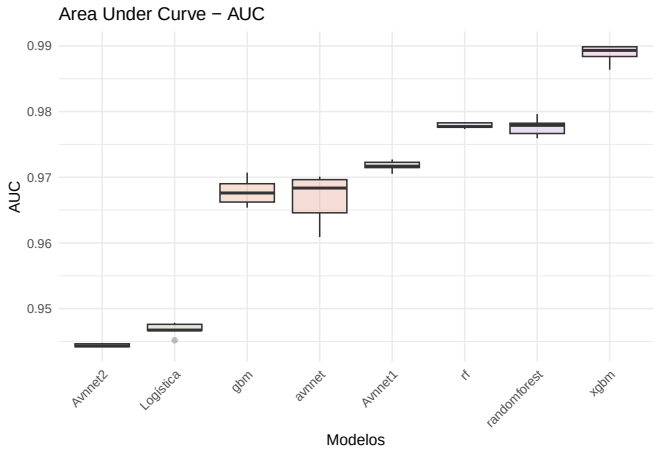

```
# Solo con las variables seleccionadas en stepwise
xgbmgrid3 <- expand.grid(
    eta = c(0.1, 0.05, 0.03, 0.01, 0.001),
    min_child_weight = c(5, 10, 20),
    nrounds = c(50,100,150,200,250,300),
    max_depth = c(2,3,4,5,6),
    gamma = 0,
    colsample_bytree = 1,
    subsample = 1)

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)
```

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:

- Pendiente, es necesario hacer feature selection?

5.6 Support Vector Machines

5.6.1 Tunning 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 optimo 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 = "cv", number = 4, savePredictions = "all")

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

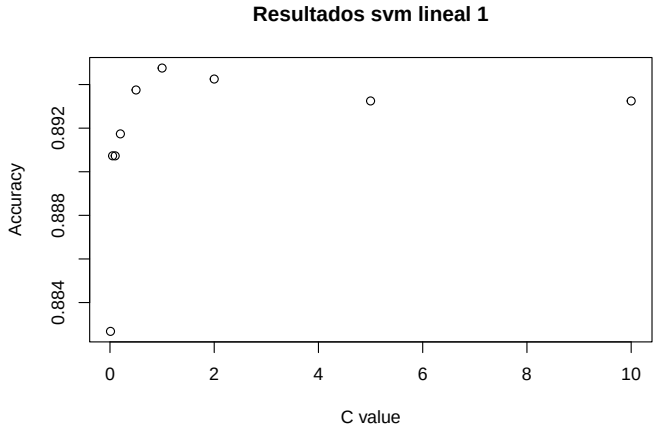
```
#svm_1
knitr::kable(svm_1$results, "pipe")
```

C	Accuracy	Kappa	AccuracySD	KappaSD
0.01	0.8826781	0.7639678	0.0019727	0.0047681
0.05	0.8907305	0.7804037	0.0061257	0.0123735
0.10	0.8907295	0.7801867	0.0073525	0.0145912
0.20	0.8917365	0.7820591	0.0071642	0.0142584
0.50	0.8937506	0.7860458	0.0069711	0.0137572
1.00	0.8947566	0.7880580	0.0082336	0.0165615
2.00	0.8942526	0.7869114	0.0090433	0.0185537
5.00	0.8932476	0.7848042	0.0076308	0.0159467
10.00	0.8932466	0.7847956	0.0084889	0.0177490

```
svm_1$bestTune
```

```
  C
6 1
```

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



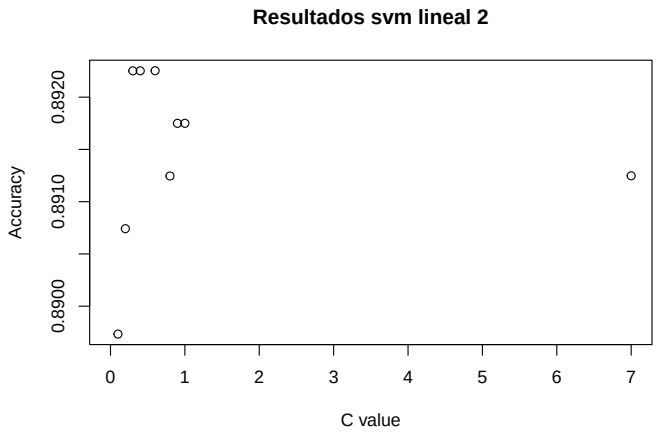
Observando bestTune y el gráfico, para este primer modelado de svm el valor C mas adecuado estaría entre 0.20 y 2 (donde se observa mejor *accuracy* en tabla), caret en bestTune indica un valor de C de 1 que es también lo que se observa en el gráfico. Con este nuevo valor de C se construye otro modelo 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	0.8897336	0.7780272	0.0091276	0.0182762
0.2	0.8907416	0.7798961	0.0099628	0.0201125
0.3	0.8922517	0.7828990	0.0089898	0.0181820
0.4	0.8922517	0.7828990	0.0089898	0.0181820
0.6	0.8922527	0.7828266	0.0094139	0.0190246
0.8	0.8912457	0.7807865	0.0090458	0.0184452
0.9	0.8917497	0.7817658	0.0092410	0.0188205
1.0	0.8917497	0.7817658	0.0092410	0.0188205
7.0	0.8912477	0.7806227	0.0106605	0.0219064

```
   C
6 0.6
```



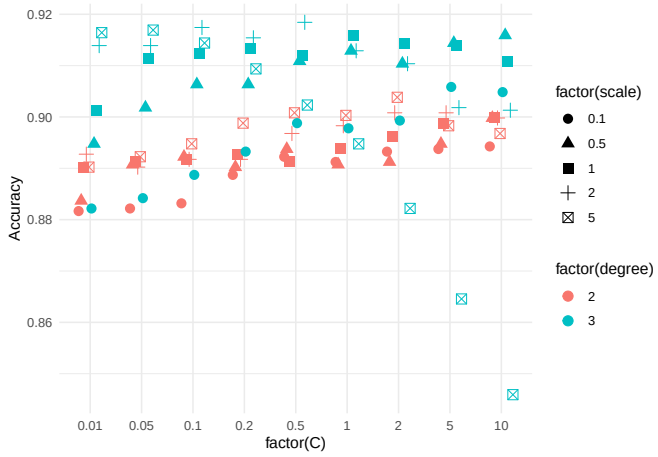
Para este dataset con SVM lineal el valor de C mas adecuado seria C=1.

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)
```

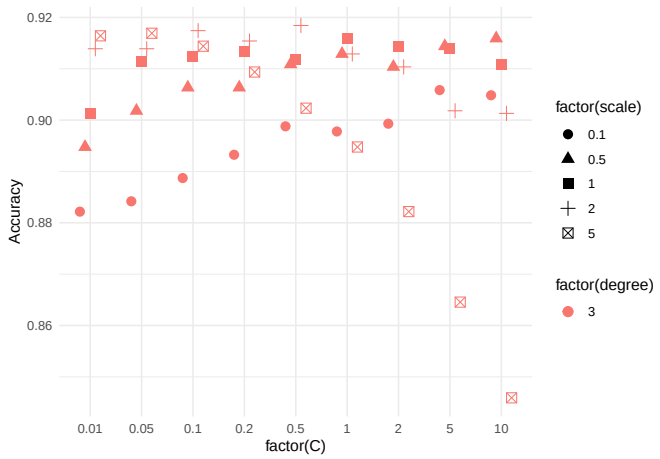
Se despliegan los resultados

	degree	scale	C
49	3	2	0.5



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:



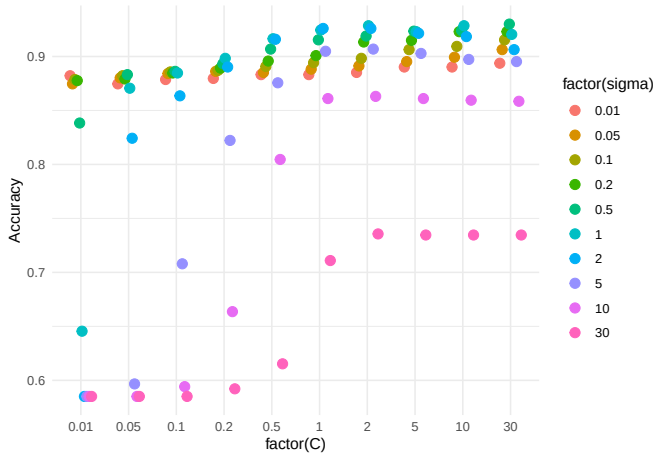
En general scale tiende a generar una curva hasta el valor C=0.5 y después descender para volver a alcanzar un accuracy alto en C=10. Como un C muy grande puede tender al sobreajuste para este dataset es mejor un C 0,5 o 1 y escala 2

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

	sigma	C
95	0.5	30



- 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

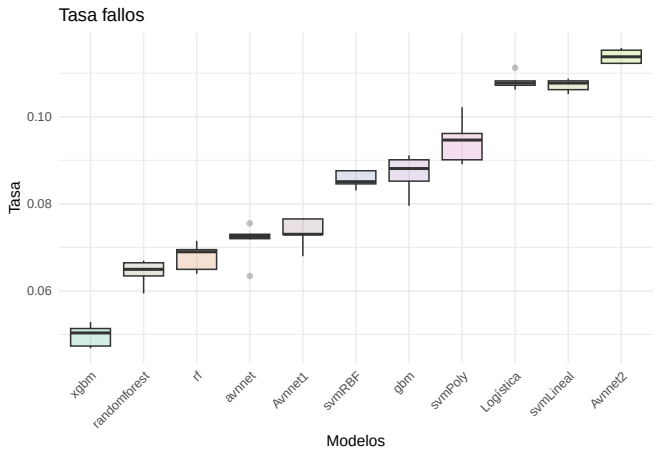
Con los valores obtenidos de svm lineal, polinomial y RBF se realiza la validación cruzada para su posterior ploteo y comparación de medias para este modelo versus los generados con los algoritmos anteriores

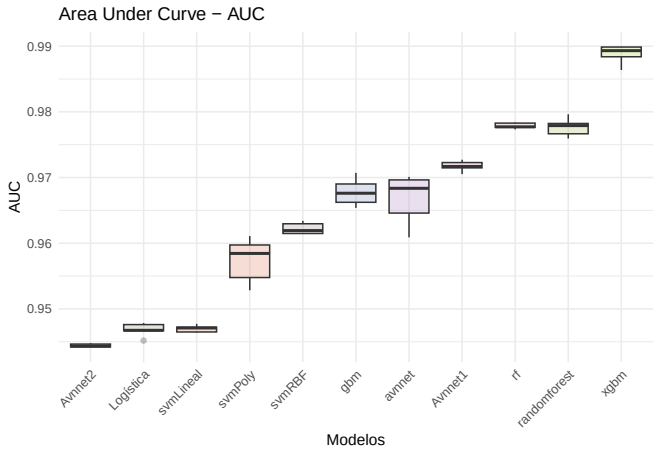
```
# 2. Validacion cruzada rep SVM ------------------------------------------
#cv para lineal
medias14 <- cruzadaSVMbin(
    data = dengue,
    vardep = "varObjBin",
    listconti = c("h10pix", "temp90", "trees", "trees90",
    "Ymin", "Ymax", "temp", "humid", "humid90"),
    listclass = c(""),
    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/MasterUCM-MachineLearningR/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 tunning) 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(
        parrf = 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 Prepapar 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
sinicio <- 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,
    sinicio = sinicio, 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<-cruzadaavnnetbin(data=archivo,
    vardep=vardep,listconti=listconti,
    listclass=listclass,grupos=grupos,sinicio=sinicio,repe=repe,
    size=c(10),decay=c(0.01),repeticiones=5,itera=200)

medias2bis<-as.data.frame(medias_2[1])
medias2bis$modelo<-"avnnet"
predi2<-as.data.frame(medias_2[2])
predi2$avnnet<-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, sinicio = sinicio, 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,
    sinicio=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, sinicio = sinicio, 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,
    sinicio = sinicio, 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 ensembled: parrf, 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: 0.9468
The fit for each individual model on the Accuracy is:
    method  Accuracy  AccuracySD
     parrf 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 tunning se construyen las nuevas medias a evaluar bajo la funcion “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,
    sinicio = sinicio, 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<-cruzadaavnnetbin(data=archivo,
    vardep=vardep,listconti=listconti,
    listclass=listclass,grupos=grupos,sinicio=sinicio,repe=repe,
    size=c(10),decay=c(0.01),repeticiones=5,itera=200)

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

# Se ha omitido el resto del codigo
```

Con la función genera graficos y una vez calculadas las medias en el proceso anterior se obtiene el grafico 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 codigo

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

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

unipredi$predi9<-(unipredi$logi+unipredi$avnnet)/2
unipredi$predi10<-(unipredi$logi+unipredi$rf)/2
unipredi$predi11<-(unipredi$logi+unipredi$gbm)/2
unipredi$predi12<-(unipredi$logi+unipredi$xgbm)/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 codigo, se construyen los promedios de tasa de fallos y AUC.

```
dput(names(unipredi))
listado<-c("logi", "avnnet",
           "rf","gbm",  "xgbm", "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 area 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")
```

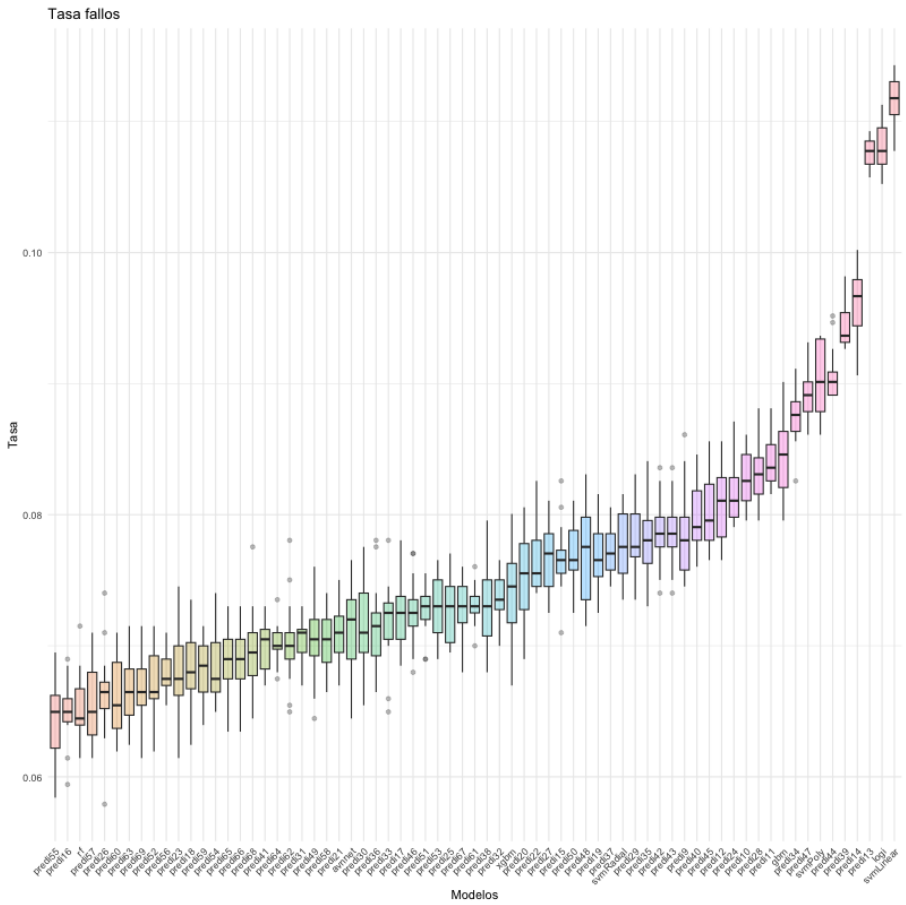
modelo	auc
predi55	0.9789378
predi57	0.9786039
predi52	0.9782842
predi60	0.9782593
predi23	0.9780586
predi69	0.9779774
predi56	0.9778908
predi26	0.9778800
predi16	0.9777369
predi54	0.9775005

En el punto siguiente se grafica para una mejor visualizacion y toma de decision

6.2.8 Paso 8, Boxplot ordenados para comparacion de medias

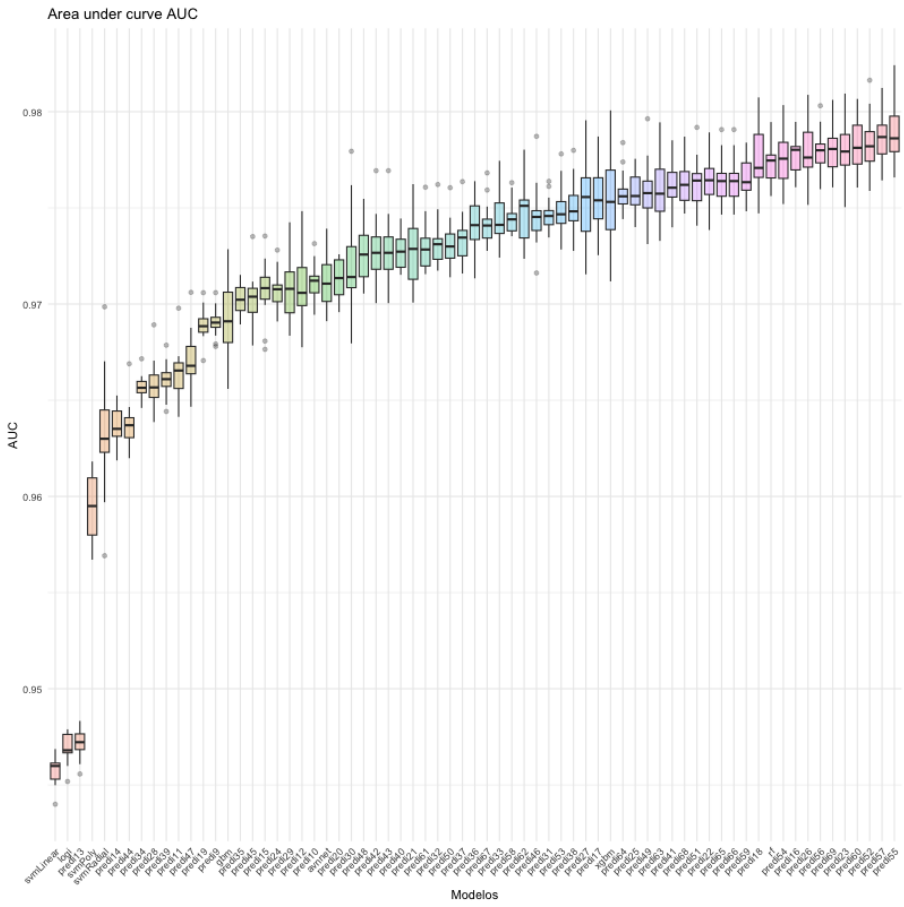
Con los resultados obtenidos de los ensamblados se generan los boxplot ordenados por tasa de fallos y auc respectivamente:

```
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")
```

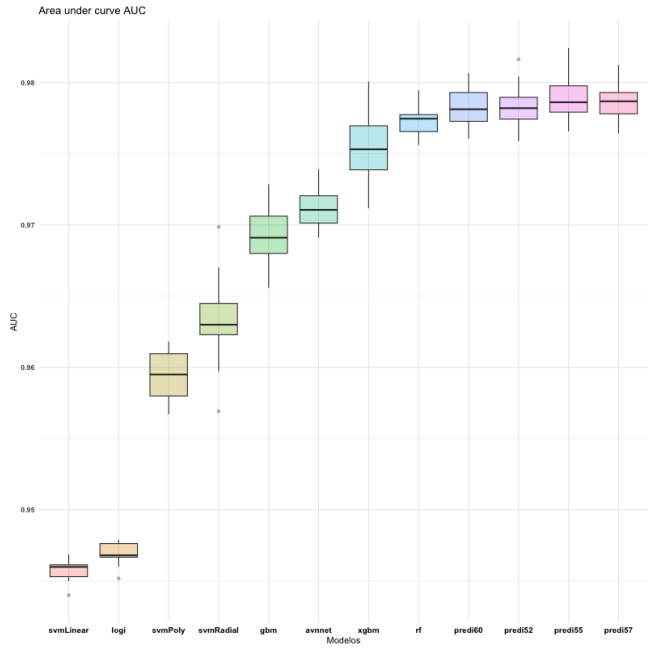


En AUC respectivamente los modelos con mayor *accuracy* son predi55, predi57 y predi52.

Como se observa en el grafico son demasiados modelos, en el apartado siguiente solo se grafican los mejores modelos de ensamblado y se comparan con los alfgoritmos 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 continuacion comparandolos con los modelos originales sin emsamblar



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 grafico 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 estaria 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 estan a la cabecera de los resultados.

7 Evaluacion del modelo ganador y conclusiones

7.1 Algoritmo a usar para este modelo

El algoritmo ganador para esta practica es random forest

7.2 Matriz de confusion para el modelo escogido

Agregar conclusiones

7.3 Sensitividad, especificidad y precision

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 ***
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
```

Comentar

7.5 Puntos de corte

```
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

          Reference
Prediction   No  Yes
       No  1020  142
       Yes   23  801

               Accuracy : 0.9169
                 95% CI : (0.9039, 0.9287)
    No Information Rate : 0.5252
    P-Value [Acc > NIR] : < 0.00000000000000022

                  Kappa : 0.8324

 Mcnemar's Test P-Value : < 0.00000000000000022

            Sensitivity : 0.8494
            Specificity : 0.9779
         Pos Pred Value : 0.9721
         Neg Pred Value : 0.8778
             Prevalence : 0.4748
         Detection Rate : 0.4033
   Detection Prevalence : 0.4149
      Balanced Accuracy : 0.9137

       'Positive' Class : Yes
```

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("Logistica", "randomforest")
contraste <- lista_medias[which(lista_medias$modelo%in%modelos),]
resultados <- t.test(contraste$tasa ~contraste$modelo)
resultados
```

```
    Welch Two Sample t-test

data:  contraste$tasa by contraste$modelo
t = 37.847, df = 10.328, p-value = 0.000000000002032
```

```
alternative hypothesis: true difference in means between group Logística and group randomforest is not equal to 0
95 percent confidence interval:
 0.04104914 0.04616134
sample estimates:
   mean in group Logística mean in group randomforest
                0.10815710                 0.06455186
```

7.7 Visualpred

Utilizando el script facilitado en los apuntes comparaciones basicas

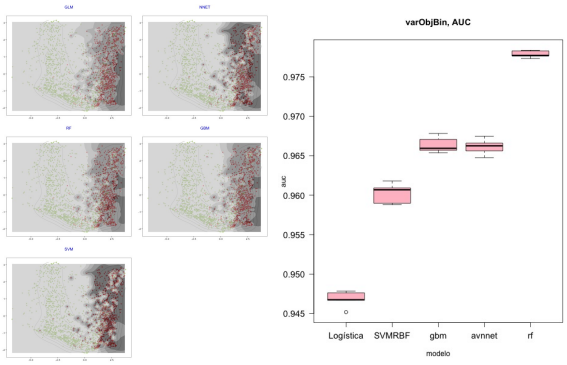


Figura 1: caption

7.8 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 18: Tasa de fallos

modelo	tasa
Avnnet1	0.0734139
Avnnet2	0.1138973
Logística	0.1081571
avnnet	0.0712991
gbm	0.0873615
randomforest	0.0645519
rf	0.0677744
svmLineal	0.1072508
svmPoly	0.0944612
svmRBF	0.0855992
xgbm	0.0497482

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

Cuadro 19: Accuracy

modelo	auc
Avnnet1	0.9717350
Avnnet2	0.9444342
Logística	0.9468125
avnnet	0.9667183
gbm	0.9678420
randomforest	0.9776175
rf	0.9778709
svmLineal	0.9469829
svmPoly	0.9573743
svmRBF	0.9622408
xgbm	0.9887692

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)
```

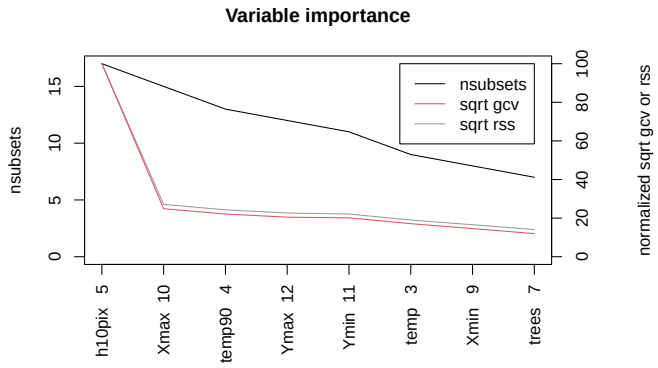
```
      humid     humid90        temp      temp90      h10pix    h10pix90
0.019011819 0.024480176 0.007135328 0.010393915 0.044306715 0.036816130
      trees     trees90        Xmin        Xmax        Ymin        Ymax
0.004198915 0.005095990 0.011027785 0.009075744 0.028311803 0.024437635
```

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)
```



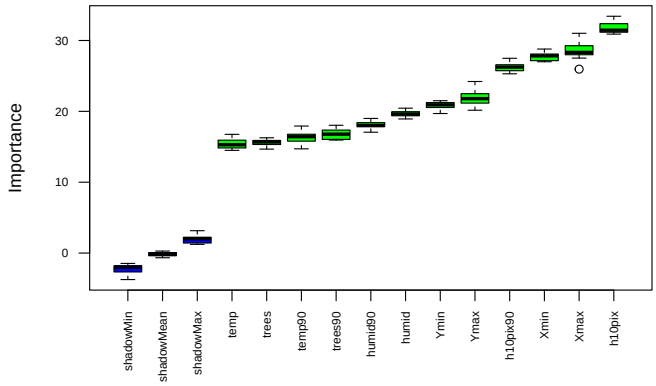
A.1.3 Utilizando la libreria Boruta

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=rfFuncs, 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 Rsquared     MAE   RMSESD RsquaredSD    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, sinicio = 1234, repe = 5)

anexo1$modelo <- "log-party"
```

```r
# Generado con MARS
anexo2 <- cruzadalogistica(
    data = data,
    vardep = "varObjBin",
    listconti = c("h10pix", "Xmax", "temp90", "Ymax", "Ymin", "temp", "Xmin",
                  "trees"),
    listclass = c(""),
    grupos = 4, sinicio = 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, sinicio = 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, sinicio = 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 Arboles de clasificacion

A.2.1 Seleccion 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*

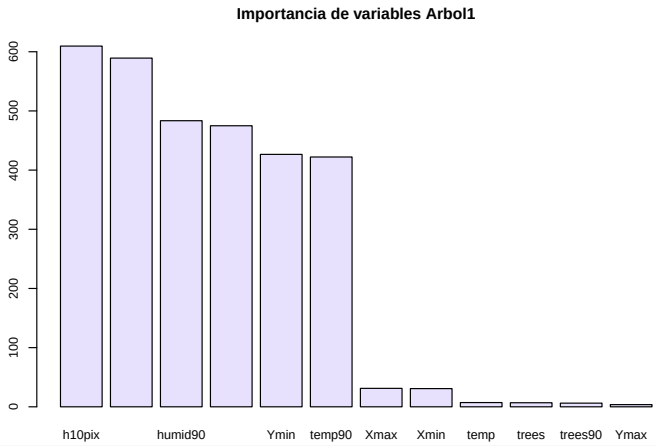
Primer modelado, modelo con todas las variables del dataset para realizar estudio de importancia de variables (que se hizo en el apartado de selección con Stepwise pero se prefiere agregarlo a la practica)

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

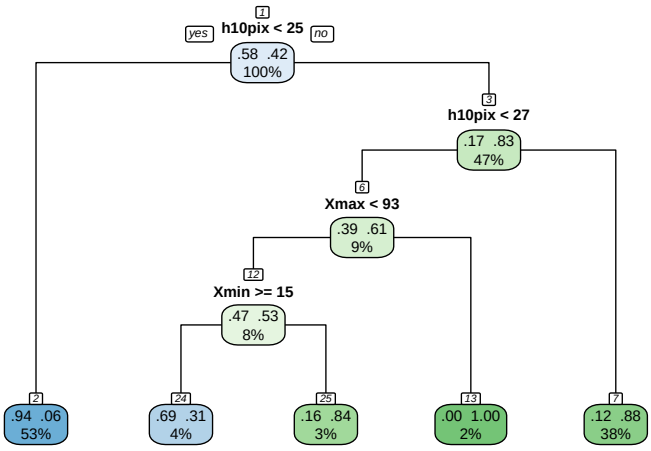
```r
# Modelo completo con gini para variable dependiente categorica
arbol1 <- rpart(factor(varObjBin) ~ .,
    data = dengue_arbol,
    minbucket = 30,
    method = "class",
    parms = list(split = "gini"))

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

Representación gráfica de las variables consideradas relevantes y árbol:

```r
# Importancia de variables
par(cex = 0.7)
barplot(height = arbol1$variable.importance, col = "#e7e1fd",
    main = "Importancia de variables Arbol1")
```




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



Observaciones:

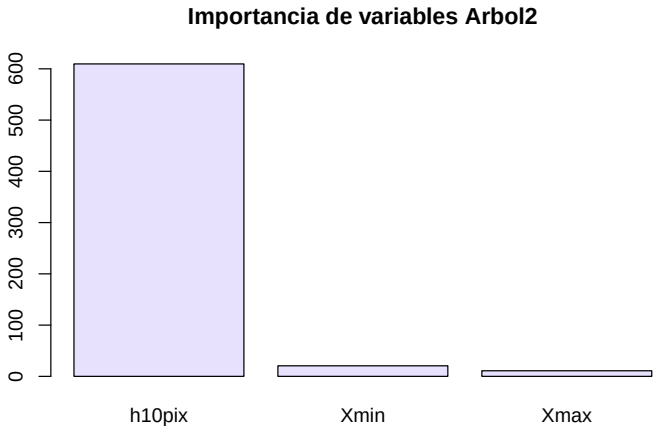
- Según el gráfico importancia de variables Árbol1, las varibles predictoras mas relevantes serian, h10pix, h10pix90, humid90, Ymin, humid e Ymax.
- agregar observaciones del árbol

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

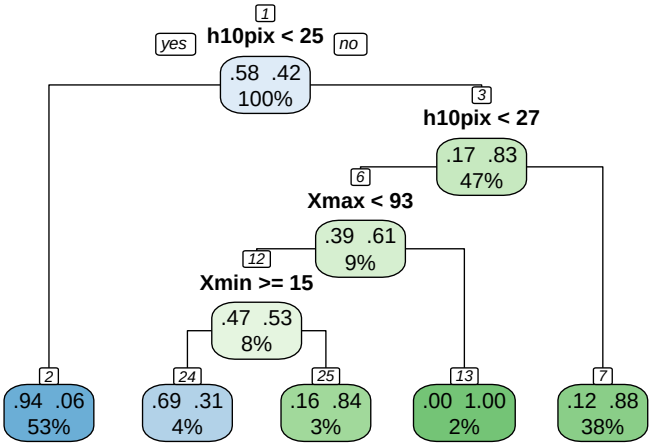
```
# Modelo con maxsurrogate = 0 para que trabaje los na's
arbol2 <- rpart(factor(varObjBin) ~ .,
    data = dengue_arbol,
    minbucket = 30,
    method = "class",
    maxsurrogate = 0,
    parms = list(split = "gini"))
```

Representación gráfica de las variables consideradas relevantes y árbol:

```
# Importancia de variables
barplot(height=arbol2$variable.importance, col = "#e7e1fd",
        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 varibles predictoras mas relevantes serian, h10pix e Ymax.
- agregar observaciones del árbol

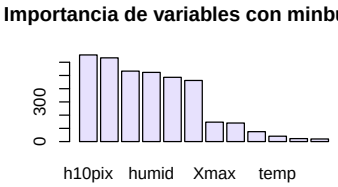
A.2.2 Tunning de algoritmos usando rpart()

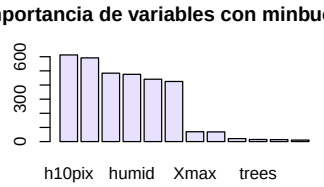
De forma inicial se realiza un primer tunning de arboles observando el comporta miento de la complejidad del árbol graficando su estructura en relación al parámetro minbucket (5, 30 y 60) utilizando la función rpart()

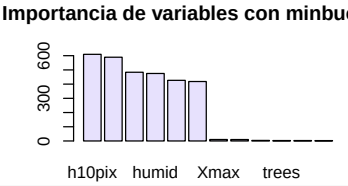
```
# Tuneado sobre complejidad del arbol con Rpart

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 tunning de minbucket

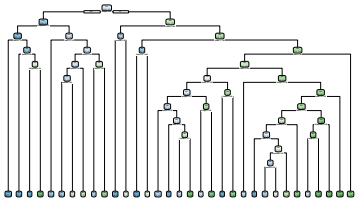
```
par(mfrow= c(2,2))
barplot(height = arbol_min5$variable.importance, col = "#e7e1fd",
        main = "Importancia de variables con minbucket 5")
barplot(height = arbol_min30$variable.importance, col = "#e7e1fd",
        main = "Importancia de variables con minbucket 30")
barplot(height = arbol_min60$variable.importance,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

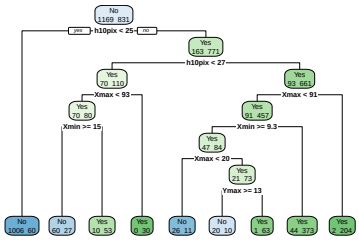


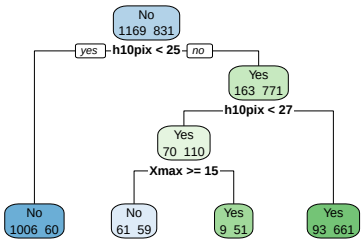





```
rpart.plot(arbol_min5, extra = 1)
rpart.plot(arbol_min30, extra = 1)
rpart.plot(arbol_min60, extra = 1)
```







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 Tunning de algoritmos usando caret()

Como indican los apuntes, no se puede realizar tunning 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
# tunning de minubucket en bucle
# Tratamiento de datos faltantes
dengue_arbol1 <- na.omit(dengue_arbol, (!is.na(dengue_arbol)))
colSums(is.na(dengue_arbol1)) > 0

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

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	0.9103816	0.9553222
10	0.8977636	0.9441889
15	0.8932587	0.9576898
20	0.9113887	0.9573321
25	0.9083675	0.9552511
30	0.8917355	0.9550814
35	0.8947678	0.9513164
40	0.9058403	0.9512422
45	0.9023131	0.9581003
50	0.9083574	0.9532965
55	0.9048383	0.9542511
60	0.9023232	0.9540407

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 cruzadaarbolbin()

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

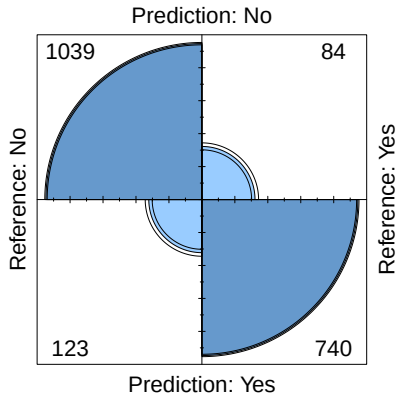
Se obtiene la matriz de confusión para este modelo

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

```
Confusion Matrix and Statistics

          Reference
Prediction   No  Yes
       No  1039   84
       Yes  123  740

               Accuracy : 0.8958
                 95% CI : (0.8815, 0.9089)
    No Information Rate : 0.5851
    P-Value [Acc > NIR] : < 0.00000000000000022

                  Kappa : 0.7868

 Mcnemar's Test P-Value : 0.008262

            Sensitivity : 0.8941
            Specificity : 0.8981
         Pos Pred Value : 0.9252
         Neg Pred Value : 0.8575
             Prevalence : 0.5851
         Detection Rate : 0.5232
   Detection Prevalence : 0.5655
      Balanced Accuracy : 0.8961
```

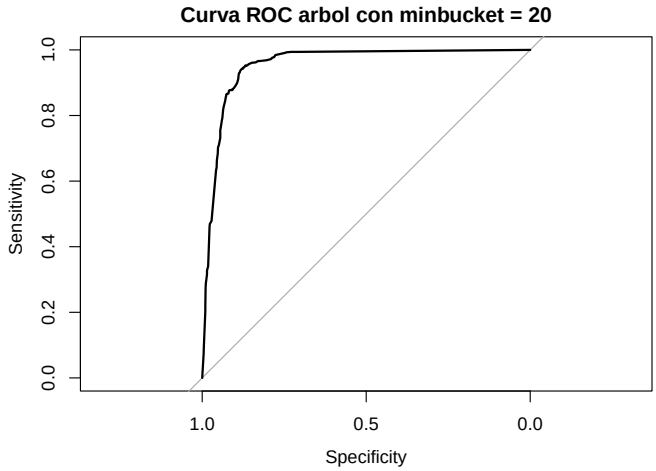
```
'Positive' Class : No
```

```
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 con minbucket = 20")
```



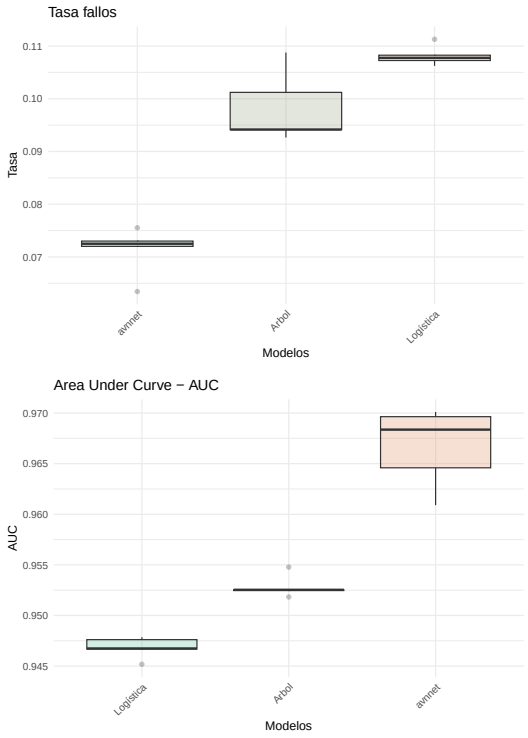
A.2.4 Comparacion con otros algoritmos de ML

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("h1Opix", "temp90", "trees", "trees90", "Ymin", "Ymax", "temp",
    "humid", "humid90"),
    listclass = c(""), grupos = 4, sinicio = 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 Libreria 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/MasterUCM-MachineLearningR/blob/master/h2o%20python/autoML_dengue.ipynb

El primer paso ha sido generar un modelo de autoML que determine cual combinacion es la mas adecuada para el set de datos dengue.

En segundo lugar y con el modelo ganador escogido anteriormente, se hace un training del modelo para comparar resultados con los obtenidos en R