

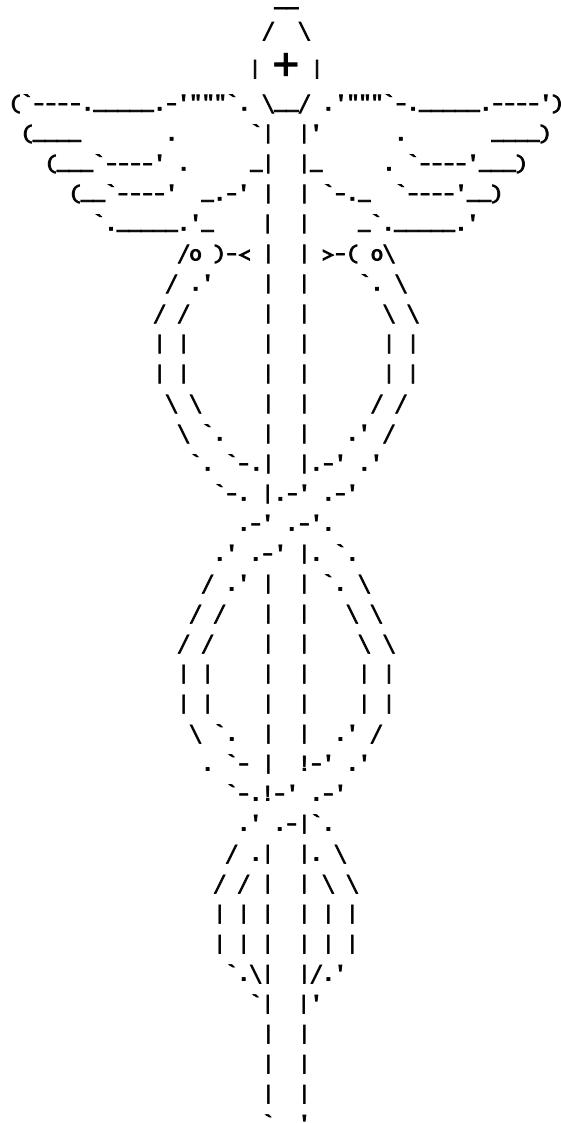
# Práctica Machine Learning

## Clasificación de pacientes con complicaciones hospitalarias

**Autor:** Fernández Hernández, Alberto

Universidad Complutense de Madrid (UCM)

20/05/2021



## Índice

<b>1. Introducción y descripción de los datos .....</b>	<b>4</b>
<b>2. Librerías empleadas .....</b>	<b>5</b>
<b>3. Depuración de los datos .....</b>	<b>6</b>
<b>3.1 Codificación a factor.....</b>	<b>7</b>
<b>3.2 Valores NA.....</b>	<b>7</b>
<b>3.3 Variables categóricas.....</b>	<b>7</b>
<b>3.4 Variables continuas.....</b>	<b>14</b>
<b>3.5 Estandarización de variables continuas .....</b>	<b>15</b>
<b>3.6 Creación de variables dummy.....</b>	<b>15</b>
<b>4. Selección de variables bajo logística.....</b>	<b>16</b>
<b>4.1 Selección de una submuestra.....</b>	<b>16</b>
<b>4.2 Stepwise AIC .....</b>	<b>17</b>
<b>4.3 Stepwise BIC .....</b>	<b>18</b>
<b>4.4 Recursive Feature Elimination (bajo logística) .....</b>	<b>19</b>
<b>4.5 Recursive Feature Elimination (bajo Random Forest) .....</b>	<b>20</b>
<b>4.6 Selección bajo logística .....</b>	<b>21</b>
<b>4.7 Tuneo y comparación final.....</b>	<b>27</b>
<b>5. Modelos iniciales con H2O .....</b>	<b>28</b>
<b>5.1 Modelo 1 .....</b>	<b>28</b>
<b>5.2 Modelo 2 .....</b>	<b>29</b>
<b>6. Redes neuronales.....</b>	<b>29</b>
<b>6.1 Modelo 1 .....</b>	<b>29</b>
<b>6.2 Modelo 2 .....</b>	<b>33</b>
<b>6.3 Comparación final .....</b>	<b>36</b>
<b>7. Bagging .....</b>	<b>37</b>
<b>7.1 Selección del número de árboles .....</b>	<b>37</b>
<b>7.2 Modelo 1 .....</b>	<b>39</b>
<b>7.3 Modelo 2 .....</b>	<b>42</b>
<b>7.4 Modelo sin reemplazamiento.....</b>	<b>45</b>
<b>7.5 Comparación final .....</b>	<b>46</b>
<b>8. Random Forest .....</b>	<b>47</b>
<b>8.1 Selección del número de árboles y mtry .....</b>	<b>47</b>
<b>8.2 Modelo 1 .....</b>	<b>51</b>

<b>8.2 Modelo 2 .....</b>	<b>53</b>
<b>8.3 Comparación final .....</b>	<b>56</b>
<b>9. Gradient Boosting .....</b>	<b>57</b>
<b>9.1 Tuneo de hiperparámetros.....</b>	<b>57</b>
<b>9.2 Comparación final .....</b>	<b>62</b>
<b>10. Support Vector Machines.....</b>	<b>63</b>
<b>10.1 SVM Lineal .....</b>	<b>63</b>
<b>10.2 SVM Polinomial.....</b>	<b>64</b>
<b>10.3 SVM RBF .....</b>	<b>66</b>
<b>10.4 Comparación modelos SVM .....</b>	<b>69</b>
<b>11. XGboost.....</b>	<b>71</b>
<b>11.1 Tuneo de hiperparámetros .....</b>	<b>71</b>
<b>11.2 Estudio Early Stopping.....</b>	<b>72</b>
<b>11.3 Tuneo max_depth .....</b>	<b>72</b>
<b>11.4 Tuneo subsample .....</b>	<b>73</b>
<b>11.5 Comparación final.....</b>	<b>74</b>
<b>12. Ensamblado .....</b>	<b>76</b>
<b>12.1 Correlación entre los modelos .....</b>	<b>76</b>
<b>12.2 Ensamblado con dos clasificadores .....</b>	<b>77</b>
<b>12.3 Ensamblado con tres clasificadores.....</b>	<b>81</b>
<b>12.4 Conclusiones ensamblado y comparación mejores algoritmos vs logística .....</b>	<b>82</b>
<b>13. Comparación con h2o.....</b>	<b>84</b>
<b>14. Probando con el dataset completo.....</b>	<b>84</b>
<b>15. Aumento del número de grupos y repeticiones .....</b>	<b>86</b>
<b>16. Variación del punto de corte .....</b>	<b>86</b>
<b>16.1 Medidas básicas (punto de corte = 0.5) .....</b>	<b>88</b>
<b>16.2 Prueba con diferentes semillas .....</b>	<b>89</b>
<b>17. Conclusiones.....</b>	<b>90</b>
<b>17.1 Mejor modelo (punto de vista computacional) .....</b>	<b>90</b>
<b>17.2 Mejor modelo (punto de corte) .....</b>	<b>91</b>
<b>17.3 Árbol básico y tabla coeficientes logística .....</b>	<b>92</b>

\* **Nota:** de cara al tuneo de hiperparámetros, se ha empleado **validación cruzada repetida de 5 grupos**, teniendo como base la semilla **1234**. En relación con el número de repeticiones, en cada modelo se comenzará con 5 y, de cara al comparación final con los mejores hiperparámetros, lo aumentamos a 10.

## 1. Introducción y descripción de los datos

El objetivo del presente proyecto consiste en **elaborar un modelo de clasificación binaria que permita predecir si un paciente presentará o será más propenso a padecer una complicación hospitalaria tras una intervención quirúrgica** <sup>1</sup>. Originalmente, el fichero (extraído de la plataforma Kaggle) contiene tres variables objetivo, dos continuas:

1. *ccsComplicationRate*: incidencia general de complicaciones hospitalarias por cada tipo de intervención quirúrgica.
2. *complication\_rsi*: índice de complicaciones hospitalarias.

Y una binaria:

3. *complication*: **indica si el paciente ha sufrido una complicación (1) o no (0)**.

Por tanto, de cara a la práctica tendremos únicamente en cuenta, como variable objetivo, la columna *complication*, descartando las dos variables continuas anteriores.

En relación con las posibles variables *input*, nos encontramos con:

### CONTINUAS

1. *bmi*: **índice de masa corporal**.
2. *Age*: **edad del paciente**.
3. *baseline\_charlson*: **índice de comorbilidad de Charlson, el cual predice la mortalidad a diez años de un paciente que puede tener una variedad de condiciones comórbidas, esto es, la presencia de uno o varios desórdenes o enfermedades**.
4. *ahrq\_ccs*: **tipo de procedimiento/intervención quirúrgica, etiquetado por la Agencia estadounidense para la Investigación Sanitaria** <sup>2</sup>.
5. *ccsMort30Rate*: **incidencia general de mortalidad a los 30 días por cada intervención (dato por el código de la columna ahrq\_ccs)**.
6. *hour*: **hora a la que se realizó la intervención**.
7. *mortality\_rsi*: **índice de estratificación de riesgo en la mortalidad a los 30 días**.

### CATEGÓRICAS

8. *asa\_status*: **estado físico del paciente establecido por la Sociedad Americana de Anestesiología** <sup>3</sup>. Contiene tres categorías:

---

<sup>1</sup> <https://www.kaggle.com/omnamahshivai/surgical-dataset-binary-classification>

<sup>2</sup> [https://www.hcup-us.ahrq.gov/toolssoftware/ccs10/CCSCategoryNames\(FullLabels\).pdf](https://www.hcup-us.ahrq.gov/toolssoftware/ccs10/CCSCategoryNames(FullLabels).pdf)

<sup>3</sup> <https://www.asahq.org/standards-and-guidelines/asa-physical-status-classification-system>

- 0: **estado I-II** (paciente sano / paciente con enfermedad sistémica leve).
  - 1: **estado III** (paciente con enfermedad sistémica grave).
  - 2: **estado IV-VI** (paciente con enfermedad muy grave / no espera sobrevivir sin la operación / muerte cerebral).
9. ***baseline\_cancer***: ¿El paciente padece algún cáncer? (1 = Si; 0 = No)
  10. ***baseline\_cvd***: ¿El paciente sufre alguna enfermedad cardio o cerebrovascular? (1 = Si; 0 = No)
  11. ***baseline\_dementia***: ¿El paciente sufre algún trastorno por demencia? (1 = Si; 0 = No)
  12. ***baseline\_diabetes***: ¿El paciente sufre diabetes? (1 = Si; 0 = No)
  13. ***baseline\_digestive***: ¿El paciente sufre alguna enfermedad gastro-intestinal? (1 = Si; 0 = No)
  14. ***baseline\_osteoart***: ¿El paciente padece osteoarthritis<sup>4</sup>? (1 = Si; 0 = No)
  15. ***baseline\_psych***: ¿El paciente padece algún desorden psiquiátrico? (1 = Si; 0 = No)
  16. ***baseline\_pulmonar***: ¿El paciente sufre alguna enfermedad pulmonar? (1 = Si; 0 = No)
  17. ***dow* o *day of week***: día de la semana en el que se realizó la intervención (0 = lunes; 1 = martes; 2 = miércoles; 3 = jueves; 4 = viernes).
  18. ***month***: mes en el que se realizó la intervención (De 0 = Enero, a 11 = Diciembre).
  19. ***moonphase***: fase lunar que tuvo lugar durante la intervención quirúrgica (0 = Luna nueva; 1 = Cuarto creciente; 2 = Luna llena; 3 = Cuarto menguante).
  20. ***mort30***: ¿El paciente presenta algún riesgo de fallecer a los 30 días? (1 = Si; 0 = No)
  21. ***gender***: Sexo del paciente (0 = Hombre; 1 = Mujer)
  22. ***race***: raza del paciente (0 = Caucásico; 1 = Afroamericano; 2 = Otro)

## 2. Librerías empleadas

A continuación, se expone un listado de las librerías empleadas en el desarrollo de la práctica:

1. ***caret***: tuneo de hiperparámetros de los diferentes algoritmos de clasificación.
2. ***data.table***: estructura de datos, similar a *data.frame*, aunque mucho más eficiente en memoria.

---

<sup>4</sup> <https://dicciomed.usal.es/palabra/osteoartritis>

3. ***ggplot2***: librería gráfica.
4. ***scorecard***: cálculo del valor de información (IV), así como el peso de la evidencia (WOE).
5. ***dummies***: transformación de variables categóricas a *dummies*.
6. ***forcats***: tratamiento de variables categóricas.
7. ***inspectdf***: librería para inspeccionar las características principales de un *dataset*, incluyendo variables categóricas, valores *missing* o distribución de las variables continuas (auto *Exploratory Data Analysis* o *auto EDA*).
8. ***dplyr***: manipulación de datos.
9. ***psych***: información general de data.frames y/o data.tables (media, asimetría, desviación típica, entre otros).
10. ***doParallel* y *parallel***: paralelización de funciones.
11. ***readxl***: lectura de ficheros *Excel* (.xlsx).
12. ***purrr***: herramientas de programación funcional.
13. ***h2o***: *auto Machine Learning* (*autoML*).
14. **Librerías y funciones proporcionadas por el profesor.**

### 3. Depuración de los datos

Inicialmente, comenzamos con la lectura del fichero .csv:

```
# Lectura del fichero
surgical_dataset <- fread("./data/Surgical-deepnet.csv", data.table = TRUE)

# Eliminamos Las dos variables objetivo continuas
surgical_dataset$ccsComplicationRate <- NULL

surgical_dataset$complication_rsi <- NULL

dim(surgical_dataset) # Filas x columnas

## [1] 14635    23
```

Nos encontramos con 14.635 observaciones, junto con las 23 variables descritas anteriormente. Si echamos un vistazo a la variable objetivo, podemos observar el desbalanceo entre ambas categorías, con apenas un 25 % de los pacientes con complicaciones (1):

```
table(surgical_dataset$complication)

##
##      0      1
## 10945  3690
```

### 3.1 Codificación a *factor*

Tras la lectura del fichero, **codificamos como *factor* tanto la variable objetivo como el resto de las variables categóricas** (para su posterior análisis):

```
# Codificamos como factor la variable objetivo...
surgical_dataset$complication <- as.factor(surgical_dataset$complication)
# ...Así como el resto de variables categoricas mencionadas anteriormente
cat_columns <- c("gender", "race", "asa_status", "baseline_cancer", "baseline_cv", 
                 "baseline_dementia", "baseline_diabetes", "baseline_digestive",
                 "baseline_osteoart", "baseline_psych", "baseline_pulmonary",
                 "dow", "month", "moonphase", "mort30")
surgical_dataset[, cat_columns] <- lapply(surgical_dataset[, cat_columns], factor)
```

A continuación, almacenamos los nombres de cada variable en un vector por separado, **en función de si es continua o categórica**:

```
# Separamos las variables en numericas, categoricas y target
# [-16] => Salvo la variable objetivo
cat_columns <- names(Filter(is.factor, surgical_dataset))[-16]
num_columns <- names(Filter(is.numeric, surgical_dataset))
target       <- "complication"
```

### 3.2 Valores NA

Como se puede comprobar a continuación, el *dataset* **no contiene valores missing en ninguna de las variables**:

```
sum(is.na(surgical_dataset))

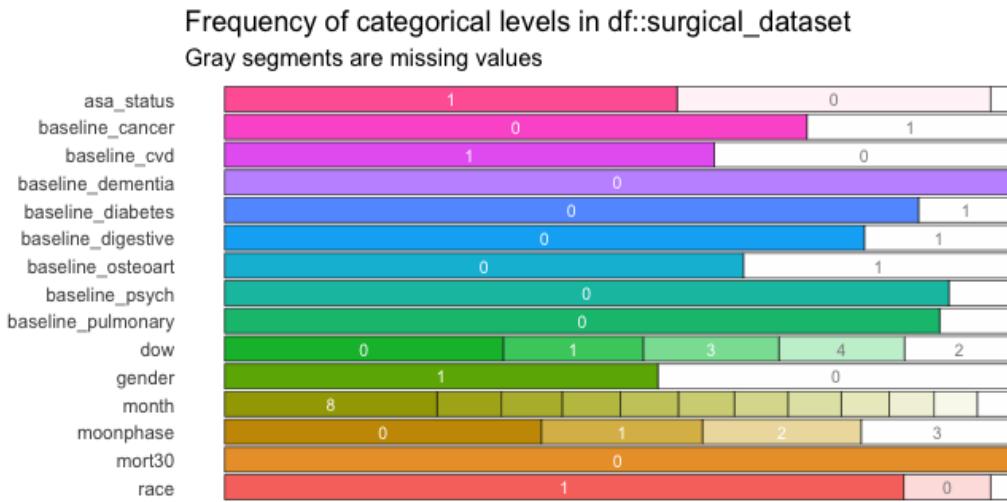
## [1] 0
```

### 3.3 Variables categóricas

Tras almacenar los nombres de cada variable, mediante la librería *inspectdf* se realizó un primer análisis exploratorio de datos automático con el que **analizar el dataset en primera instancia**. Dado que el contenido del informe es muy extenso, se incluirá en la memoria el contenido esencial (**el informe completo se incluye, desglosado, en los anexos *EDA\_report.pdf* y *WOEBIN\_factor\_variables.pdf***).

Sobre dicho informe, **comenzamos remarcando la frecuencia de aparición de los niveles de cada variable categórica**:

```
x <- inspect_cat(surgical_dataset[, cat_columns], include_int = TRUE)
show_plot(x)
```



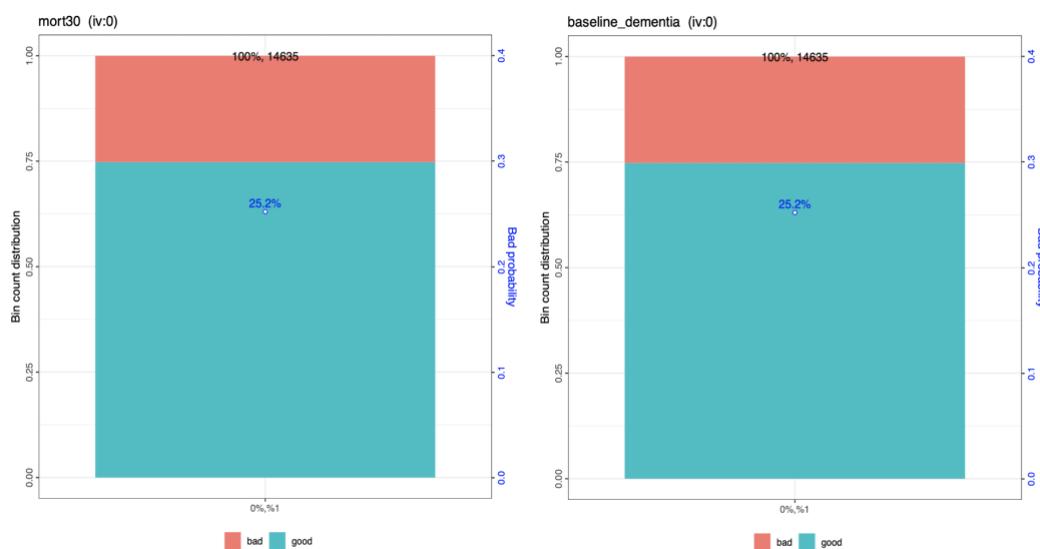
**Figure 1. Frecuencias variables categóricas (inspectdf)**

A simple vista, prácticamente todas las categorías presentan una alta frecuencia de aparición, **salvo por *baseline\_dementia* y *mort30***, donde el número de observaciones a 1 es de 71 y 58:

```
surgical_dataset[, c("baseline_dementia", "mort30")] %>% map(table)

## $baseline_dementia      $mort30
## 
##      0      1              0      1
## 14564    71        14577    58
```

Es decir, **se tratan de variables con pocas observaciones con valor 1**. De hecho, si analizamos el valor de información, haciendo uso del paquete *scorecard*, el cual nos permite estudiar el “poder predictivo de una variable”, observamos que el valor de información es cero, **dada la poca representatividad de los valores a 1**, de forma que la librería acaba uniendo ambas categorías, lo que se traduce en un escaso poder de predicción:



**Figure 2. Mort\_30 y Baseline\_dementia (IV)**

Por otro lado, si analizamos la proporción de aparición de la variable objetivo sobre cada categoría:

```
##-- baseline_dementia
surgical_dataset %>%
  count(baseline_dementia, complication) %>%
  group_by(complication)

## # A tibble: 4 x 3
## # Groups: complication [2]
##   baseline_dementia complication     n
##   <fct>           <fct>       <int>
## 1 0               0             10913
## 2 0               1             3651
## 3 1               0              32
## 4 1               1              39

##-- mort30
surgical_dataset %>%
  count(mort30, complication) %>%
  group_by(complication)

## # A tibble: 4 x 3
## # Groups: complication [2]
##   mort30 complication     n
##   <fct> <fct>       <int>
## 1 0      0             10924
## 2 0      1             3653
## 3 1      0              21
## 4 1      1              37
```

A simple vista, en ambas variables **no existe una clara separación sobre la variable objetivo**. Por tanto, se ha tomado la decisión de descartar ambas columnas del conjunto de datos:

```
surgical_dataset$baseline_dementia <- NULL; surgical_dataset$mort30 <- NULL

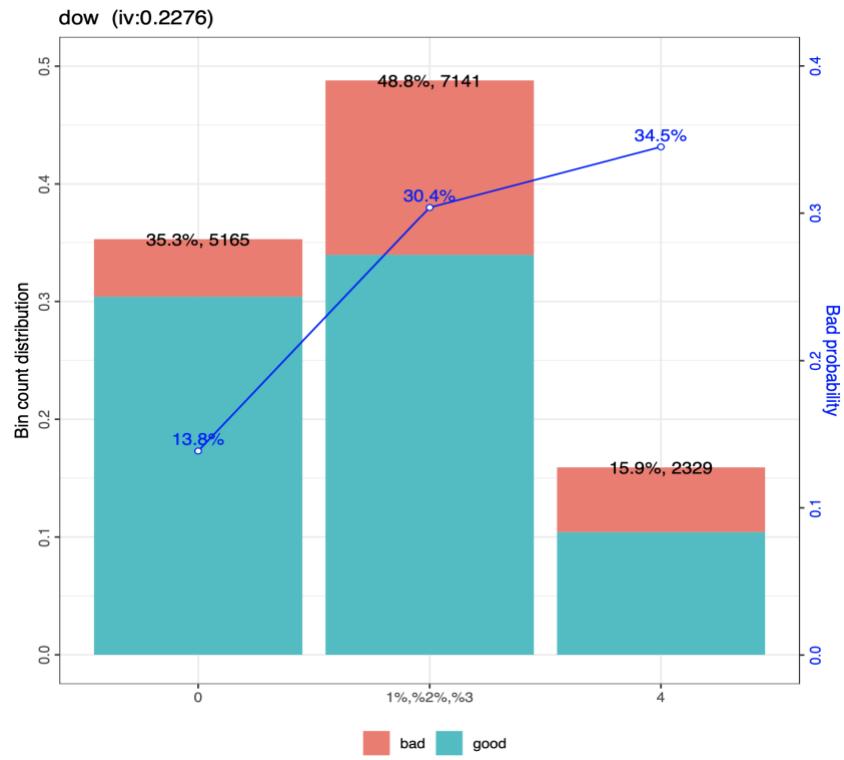
# Actualizamos el vector de las variables categoricas
cat_columns <- setdiff(cat_columns, c("baseline_dementia", "mort30"))
```

### 3.3.1 Agrupación de variables categóricas

Por otro lado, nos encontramos con dos variables cuyas categorías pueden ser agrupadas, según la información proporcionada por el paquete *scorecard*:

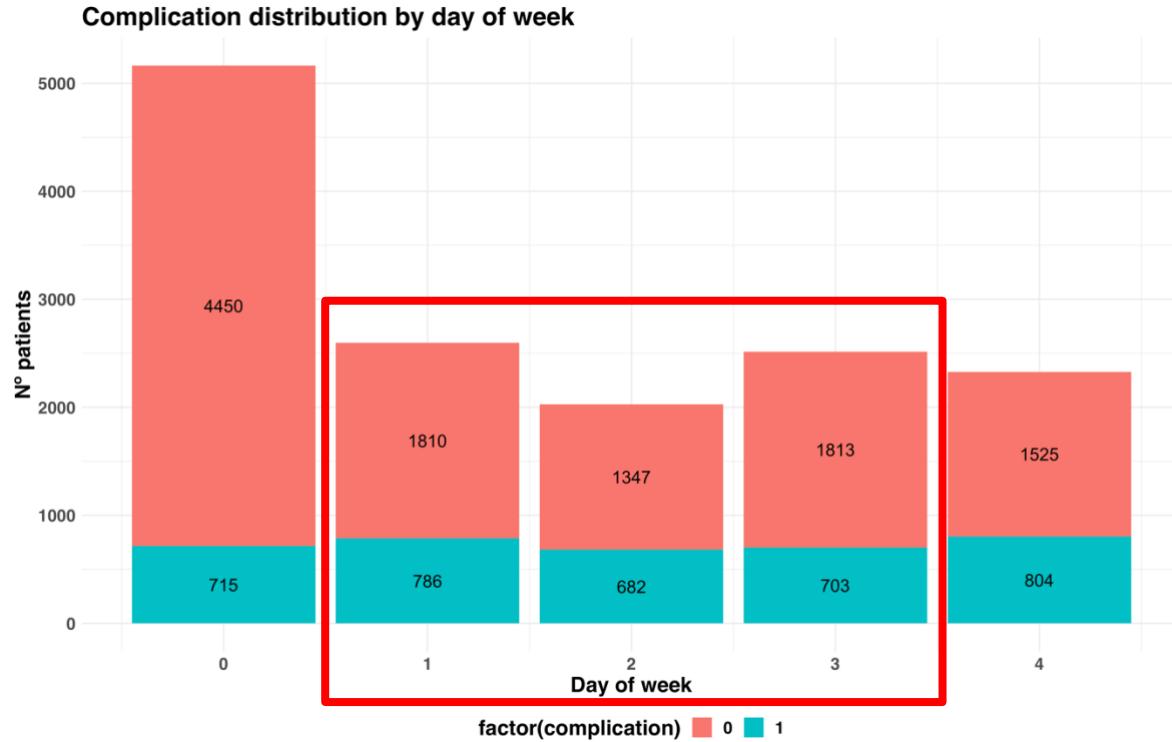
#### DÍA DE LA SEMANA (dow):

Sobre dicha variable, **observamos una relación "lineal" en la proporción de aparición de la variable objetivo a lo largo de los diferentes días de la semana**, comenzando por el lunes (0), con el menor porcentaje de complicaciones hospitalarias (alrededor del 14 %), seguido de los martes-miércoles-jueves, donde el porcentaje aumenta hasta el 30 %, y finalizando con los viernes, donde se alcanza el mayor porcentaje de complicaciones hospitalarias: 34.5 %:



*Figure 3. Día de la semana o dow (IV)*

Por otro lado, si analizamos detenidamente el gráfico de distribución con *ggplot*:



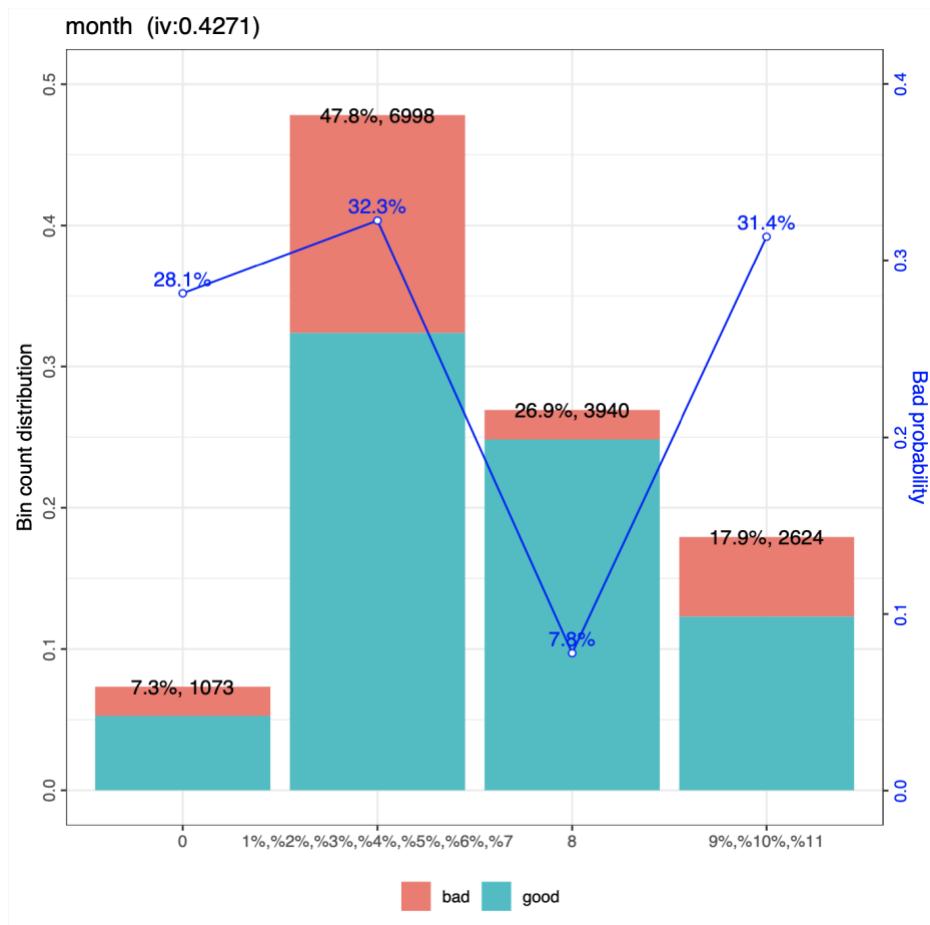
*Figure 4. Distribución de "complication" sobre el día de la semana (dow)*

Observamos que la proporción de aparición de pacientes con complicaciones es muy similar entre los martes, miércoles y jueves:

	dow	sin.comp	con.comp	total	prop.compliacion
## 1	1	4450	715	2596	30.3
## 2	2	1810	786	2029	33.6
## 3	3	1347	682	2516	27.9
<b>## 4 En conjunto (1-2-3)</b>	<b>4970</b>	<b>2171</b>	<b>7141</b>		<b>30.4</b>
## 5	4	1525	804	2329	34.5

En conjunto, **acumulan alrededor del 30 % de pacientes con complicaciones**, mientras que con tan solo el viernes aumenta hasta alcanzar el 34 %. Por tanto, dado que los martes, miércoles y jueves presentan una proporción de aparición similar, **las agrupamos en torno a una misma categoría**:

1. **Lunes (0)**
2. **Martes-Miercoles-Jueves (1-3)**
3. **Viernes (4)**



*Figure 5. Month (IV)*

## MES (month):

En este caso, llaman la atención tres grandes grupos: en primer lugar el mes de enero (0), con una proporción del 28.1 %, **seguido de los meses de febrero (1) hasta agosto (7) con una proporción ligeramente superior de pacientes con complicaciones (32 %)**, similar que prácticamente cualquiera de los siguientes 7 meses. Por el contrario, **durante el mes de septiembre (8) el porcentaje se desploma hasta el 7.8 %**, porcentaje que vuelve a aumentar en los tres meses siguientes (octubre, noviembre y diciembre), hasta el 31 %.

De hecho, si analizamos el gráfico de distribución:

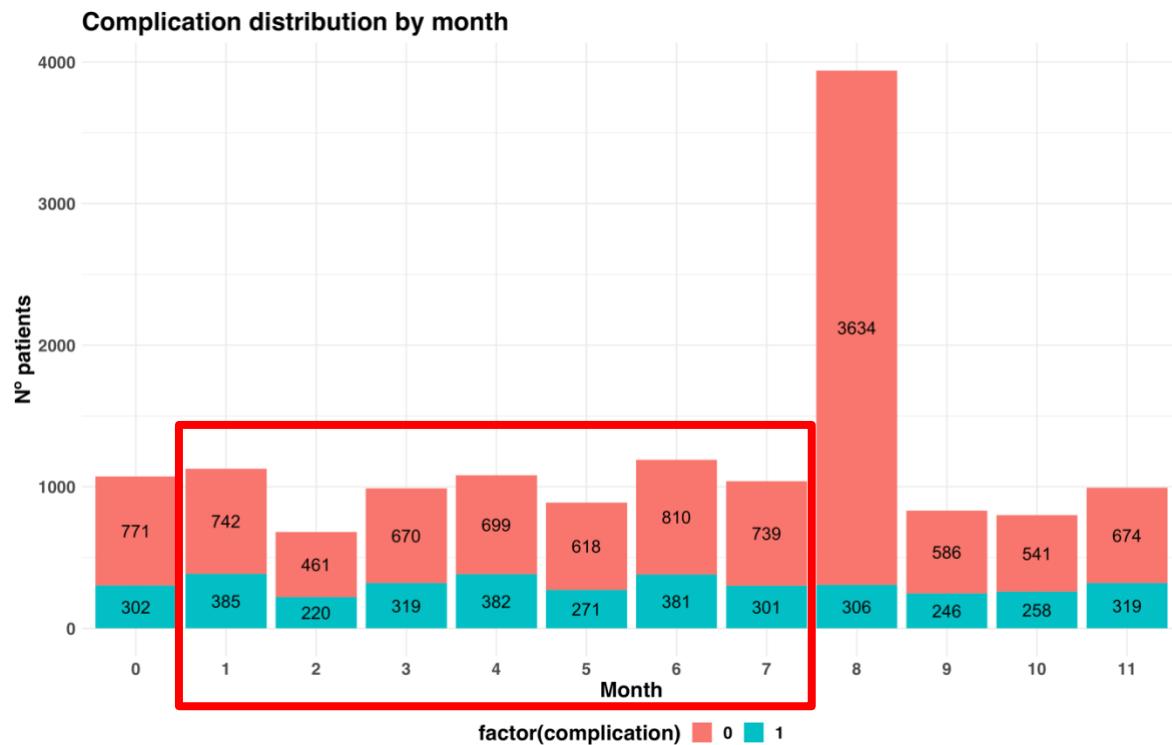


Figure 6. Gráfico distribución "complication" sobre la variable "month"

Sucede un comportamiento similar al de la variable *dow*: salvo el mes de septiembre, **la proporción de pacientes con complicaciones en prácticamente todos los meses es muy similar, de forma que podemos agrupar varios de los meses en una misma categoría, tal y como hemos comprobado anteriormente**. A modo de ejemplo, analicemos la proporción en el número de complicaciones hospitalarias de cada mes por separado y en conjunto:

##	dow	sin.comp	con.comp	total	prop.complicacion
## 1	0	771	302	1073	28.1
## 2	1	742	385	1127	34.2
## 3	2	461	220	681	32.3
## 4	3	670	319	989	32.3
## 5	4	699	382	1081	35.3
## 6	5	618	271	889	30.5
## 7	6	810	381	1191	32.0
## 8	7	739	301	1040	28.9
## 9	Uniendo 1-7	4739	2259	6998	32.3

## 10	8	3634	306	3940	<b>7.8</b>
## 11	9	586	246	832	<b>29.6</b>
## 12	10	541	258	799	<b>32.3</b>
## 13	11	674	319	993	<b>32.1</b>
<b>## 14 Uniendo 9-11</b>	<b>1801</b>	<b>823</b>	<b>2624</b>		<b>31.4</b>

Como podemos comprobar, la proporción de pacientes con complicaciones hospitalarias, entre los meses de febrero (1) y agosto (7), se sitúa en torno al 32 % si agrupamos dichas categorías. Del mismo modo, los meses de octubre, noviembre y diciembre (9, 10 y 11) se sitúan en torno al 31 %. Como consecuencia, y dado que dichas categorías presentan una proporción similar en cuanto a pacientes con complicaciones se refiere, las agrupamos:

1. **Enero (0)**
2. **Febrero a Agosto (1-7)**
3. **Septiembre (8)**
4. **Octubre, Noviembre y Diciembre (9-10-11)**

En relación con el resto de las variables categóricas, si analizamos sus correspondientes valores de información (iv):

```
##           variable    iv
## 1 baseline_osteoart 0.5246
## 2                 month 0.4271
## 3                  dow 0.2276
## 4             moonphase 0.2119
## 5 baseline_cancer 0.1358
## 6 baseline_cvd 0.0429
## 7            gender 0.0221
## 8 baseline_digestive 0.0134
## 9      asa_status 0.0062
## 10 baseline_pulmonary 0.0053
## 11 baseline_diabetes 0.0013
## 12            race 0.0002
## 13 baseline_psych 0.0001
```

Por lo general, la mayoría de las variables categóricas presentan, como primera impresión, un buen poder predictivo, con ciertas excepciones como *asa\_status*, *baseline\_pulmonary*, *baseline\_diabetes*, *race* o *baseline\_pysch*, cuyo IV no alcanza el 0.01 (como normal general, un valor de información inferior a 0.1-0.02 se traduce en un poder predictivo muy bajo o prácticamente nulo)<sup>5</sup>.

De este modo, de cara a la selección de variables, en caso de ser necesario descartar alguna variable categórica, empleamos esta tabla como referencia.

---

<sup>5</sup> [https://docs.tibco.com/pub/sfire-dsc/6.5.0/doc/html/TIB\\_sfire-dsc\\_user-guide/GUID-07A78308-525A-406F-8221-9281F4E9D7CF.html](https://docs.tibco.com/pub/sfire-dsc/6.5.0/doc/html/TIB_sfire-dsc_user-guide/GUID-07A78308-525A-406F-8221-9281F4E9D7CF.html)

### 3.4 Variables continuas

En relación con las variables continuas, nos encontramos con algunos casos en los que el número de valores únicos es bastante reducido:

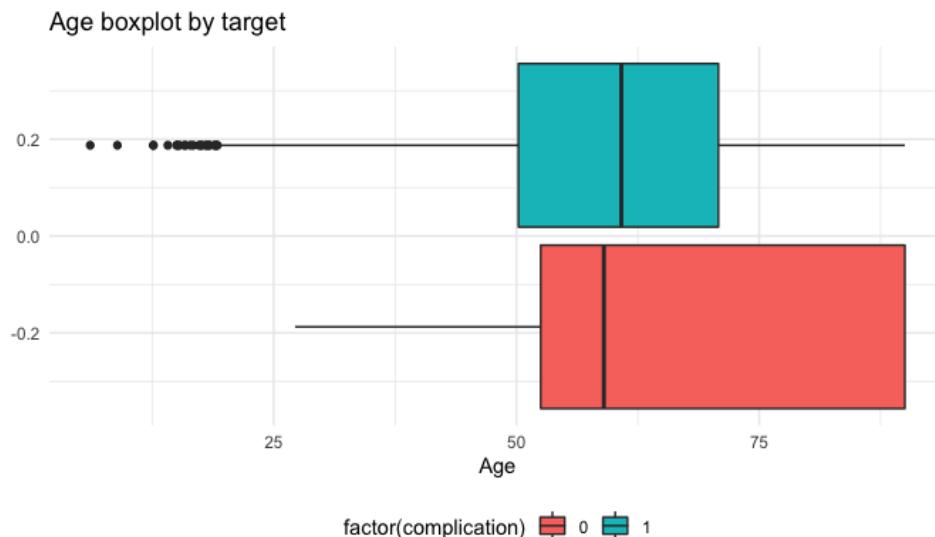
```
#-- Nº Valores únicos (continuas)
apply(surgical_dataset[, num_columns], 2, function(x) {length(unique(x)))}

##          bmi           Age baseline_charlson      ahrq_ccs
##      3095          672                  14             22
##  ccsMort30Rate    hour     mortality_rsi
##          20          725                 633
```

A modo de ejemplo, variables como *baseline\_charlson*, *ahrq\_ccs* o *ccsMort30Rate* presentan 14, 22 o 20 valores únicos, respectivamente. **No obstante, se tomó la decisión de mantener dichas variables como continuas**, principalmente para facilitar la selección de variables, evitando con ello crear nuevas variables *dummy* (una por cada categoría), utilizando con ello menos parámetros para describir el conjunto de datos.

Por otro lado, en un primer análisis exploratorio se observó que en las variables continuas, por lo general, **no existe una clara separación lineal entre ambas variables objetivo**. A modo de ejemplo, observemos la distribución de las variables *age* y *bmi*:

**AGE:**



*Figure 7. Distribución de la edad (Age) sobre la variable objetivo*

En este caso, **bien es cierto que los pacientes que no sufren complicaciones oscilan entre los 25 y los 90 años, aproximadamente, mientras que existe un número de pacientes (menores a 25 años), donde padecen alguna complicación hospitalaria**. Pese a ello, la separación no es tan clara, dada , la amplitud que presentan las "cajas ".

## BMI:

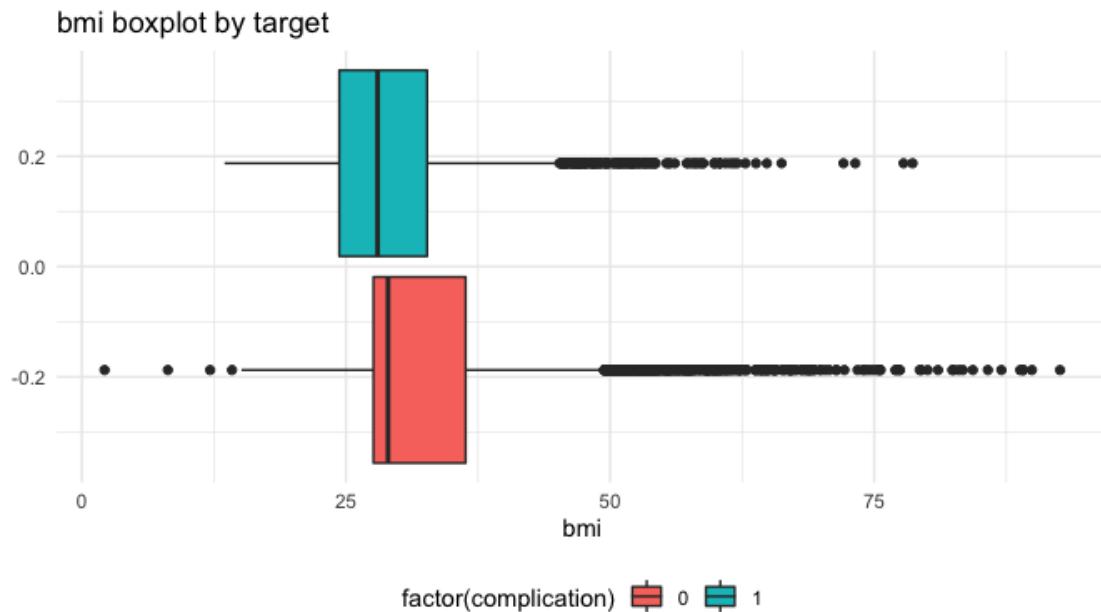


Figure 8. Distribución del índice de masa corporal (bmi) sobre la variable objetivo

Incluso con el índice de masa de corporal, a simple vista no existe una clara separación que diferencie a ambas variables objetivo en función de su valor.

## 3.5 Estandarización de variables continuas

Una vez realizado el primer análisis, procedemos a la estandarización de las variables continuas:

```
# --- Estandarizacion de variables
surgical_dataset_stnd <- surgical_dataset
# media
media      <- sapply(surgical_dataset_stnd[, num_columns], mean)
# sd
desv.tipica <- sapply(surgical_dataset_stnd[, num_columns], sd)
surgical_dataset_stnd[, num_columns]<-scale(surgical_dataset_stnd[,num_columns],
                                              center = media,
                                              scale = desv.tipica)
```

## 3.6 Creación de variables *dummy*

A continuación, convertimos las variables categóricas (con más de una categoría) en variables *dummy*, mediante la función *dummy.data.frame*, convirtiendo de este modo todas las variables a formato numérico:

```
columnas_dummy <- c("asa_status", "dow", "month", "moonphase", "race")
surgical_dataset_stnd_dummy <- dummy.data.frame(surgical_dataset_stnd[, columnas
_dummy], sep = ".")
```

Una vez codificadas, **comprobamos si existen variables *dummy* con una frecuencia de aparición menor a 100**, como referencia:

```
names(surgical_dataset_stnd_dummy[, colSums(surgical_dataset_stnd_dummy==0) < 100,  
drop = FALSE])  
  
## character(0)
```

Como podemos comprobar, todas las variables *dummy* codificadas presentan una frecuencia superior. A continuación, **unimos en un único *data.table* tanto las variables numéricas, las variables categóricas (binarias), las variables *dummy*, así como la variable objetivo**:

```
surgical_dataset_final <- cbind(  
  surgical_dataset_stnd[, num_columns],  
  surgical_dataset_stnd[, cat_columns[!cat_columns %in% columnas_dummy]],  
  surgical_dataset_stnd_dummy,  
  surgical_dataset_stnd[, target]  
)
```

Finalmente, de cara a la elaboración de los modelos **codificamos la variable objetivo como "Yes" / "No"**, renombrando la variable como *target*:

```
## Renombramos la variable objetivo como "target"  
names(surgical_dataset_final)[33] <- "target"  
  
# Renombramos las columnas para adecuarlas a formulas  
names(surgical_dataset_final) <- make.names(names(surgical_dataset_final))  
  
# 1 - "Yes" ; 0 - "No"  
surgical_dataset_final$target <- ifelse(  
  surgical_dataset_final$target == 1,  
  "Yes",  
  "No"  
)  
  
surgical_dataset_final$target <- as.factor(surgical_dataset_final$target)  
## Numero de variables finales (dummies incluidas): 32
```

## 4. Selección de variables bajo logística

### 4.1 Selección de una submuestra

Durante la selección de variables, e incluso con determinados modelos como redes neuronales o SVM, el tiempo de cómputo que requería era demasiado elevado. Como consecuencia, se planteó utilizar, en lugar del *dataset* completo con 14.000 observaciones, **un subconjunto de menor tamaño**, agilizando de este modo el cálculo.

Dado que la variable objetivo se encuentra desbalanceada, como pudimos comprobar en el apartado de Depuración, recurrimos al **muestreo estratificado**, el cual nos asegura la proporción de la variable objetivo. Para ello, *caret* dispone de la función *createDataPartition*,

con el que obtener una submuestra estratificada de forma aleatoria. Concretamente, para el desarrollo de la práctica empleamos un subconjunto con el **40 % de las observaciones**:

```
##-- Muestreo estratificado (aleatorio)
set.seed(1234)
partitions <- createDataPartition(surgical_dataset_final$target, p = 0.40, list
= FALSE)

surgical_dataset_final_est <- surgical_dataset_final[partitions, ]

# Mantenemos la misma proporcion en la variable objetivo
table(surgical_dataset_final_est$target)

##
##   No   Yes
## 4378 1476
```

No obstante, elegir un 40 % de los datos resultar ser una decisión bastante azarosa **¿Es suficiente? Más importante aún ¿Se obtienen los mismos resultados o similares?** Si nos fijamos en la depuración final, nos encontramos con  $14.635 / 32 \sim 457$  observaciones por variable, aproximadamente. Sin embargo, con tan solo el 40 %, obtendríamos  $(14.635 \times 0.4) / 32 \sim 183$  observaciones por variable. Es decir, con una menor proporción de observaciones por variable, es posible que los modelos no sean exactamente iguales con ambos conjuntos, pudiendo llegar incluso a perder precisión.

Por ello, durante la selección de variables **se tomó la decisión de aplicar las mismas técnicas sobre ambos conjuntos**:

1. *stepwise AIC*
2. *stepwise BIC*
3. *RFE o Recursive Feature Elimination (sobre una regresión logística)*
4. *RFE (sobre Random Forest)*

A continuación, y en caso de obtener una selección de variables similar, **aplicamos un modelo de validación cruzada logística en ambos casos, comprobando de este modo si los resultados son similares**, tanto con el *dataset* original como con tan solo el 40 % de las observaciones. En caso de que las estadísticas sean similares, de cara al resto de modelos trabajamos con el subconjunto.

## 4.2 Stepwise AIC

Comenzamos con el mejor *set* de variables obtenido por *stepwise AIC*:

```
##-- Stepwise AIC (100 semillas y 0.8 x 0.2 train-test)
lista.variables.aic <- steprepetidobinaria(data=surgical_dataset,
                                              vardep=target, listconti=vars,
                                              sinicio=1234, sfinal=1334,
                                              porcen=0.8, criterio="AIC")
tabla.aic <- lista.variables.aic[[1]]
```

```

##      n variable_dataset_original   variable_subset
## 1      1               mortality_rsi    mortality_rsi
## 2      2             ccsMort30Rate   ccsMort30Rate
## 3      3                  bmi          bmi
## 4      4            month.8       month.8
## 5      5        baseline_cvd    baseline_cvd
## 6      6                  dow.0      dow.0
## 7      7                  Age          Age
## 8      8           moonphase.0   moonphase.0
## 9      9           month.0       month.0
## 10     10         asa_status.0   asa_status.0
## 11     11 baseline_osteoart baseline_osteoart
## 12     12 baseline_charlson  baseline_charlson
## 13     13           ahrq_ccs      ahrq_ccs
## 14     14 baseline_diabetes baseline_diabetes
## 15     15   baseline_cancer      -
## 16     16   baseline_psych      -
## 17     17      asa_status.1      -

```

Como podemos observar, la diferencia entre el *dataset* original y la submuestra es de tan solo 3 variables: *baseline\_cancer*, *baseline\_psych* y *asa\_status.1*. Además, si analizamos las frecuencias de aparición de los cuatro mejores *sets*:

##	set	nº	var	freq_variable_subset		nº	var	freq_variable_dataset_original
##		<b>1</b>	<b>14</b>		<b>8</b>		<b>17</b>	<b>7</b>
##		2	13		5		15	4
##		3	15		4		16	4
##		4	14		4		18	3

En cualquiera de ellos, el número de variables en los mejores *sets* es similar: en torno a 13-15 en el *subconjunto* y 15-18 *features* en el *dataset* original.

### 4.3 Stepwise BIC

Por otro lado, analicemos la selección de variables bajo *stepwise BIC*:

```

-- Stepwise BIC (100 repeticiones)
lista.variables.bic <- steprepetidobinaria(data=surgical_dataset,
                                              vardep=target, listconti=vars,
                                              sinicio=1234, sfinal=1334,
                                              porcen=0.8, criterio="BIC")
tabla.bic <- lista.variables.bic[[1]]

##      n variable_dataset_original   variable_subset
## 1      1               mortality_rsi    mortality_rsi
## 2      2             ccsMort30Rate   ccsMort30Rate
## 3      3                  bmi          bmi
## 4      4            month.8       month.8
## 5      5                  dow.0      dow.0
## 6      6                  Age          Age
## 7      7           moonphase.0   moonphase.0
## 8      8        baseline_osteoart baseline_osteoart
## 9      9           asa_status.0   asa_status.0
## 10     10      baseline_cancer      -

```

En este caso, la diferencia en ambas selecciones es de tan solo una única variable: *baseline\_cancer*. Nuevamente, si analizamos las frecuencias de aparición de los cuatro mejores *sets*:

	set	nº	var	freq_variable_subset		nº	var	freq_variable_dataset_original	
##	1	9		8		10		53	
##	2	9		6		10		28	
##	3	9		4		6		11	
##	4	8		4		5		9	

En el caso del *subset*, el número de variables es muy similar (entre 8 y 9). Sin embargo, **llama la atención la elevada frecuencia que presenta el primer set obtenido con el dataset original: 53 frente a 28, 11 o 9 del resto de selecciones candidatas**, donde además recordemos que la única diferencia entre los dos primeros *sets* es de tan solo una variable: *baseline\_cancer*. ¿Influirá de cara a los resultados de un modelo logístico?

#### 4.4 Recursive Feature Elimination (bajo logística)

Antes de comparar ambas selecciones *stepwise*, debemos recalcar un problema: en ambos métodos de selección de variables, y con ambos *datasets*, **el número de parámetros escogidos es bastante elevado**, especialmente con *AIC*. Como consecuencia, a los métodos anteriores añadimos una selección de variables adicional: *Recursive Feature Elimination*, tanto bajo logística como con *Random Forest*, eliminando las variables más débiles de forma recursiva:

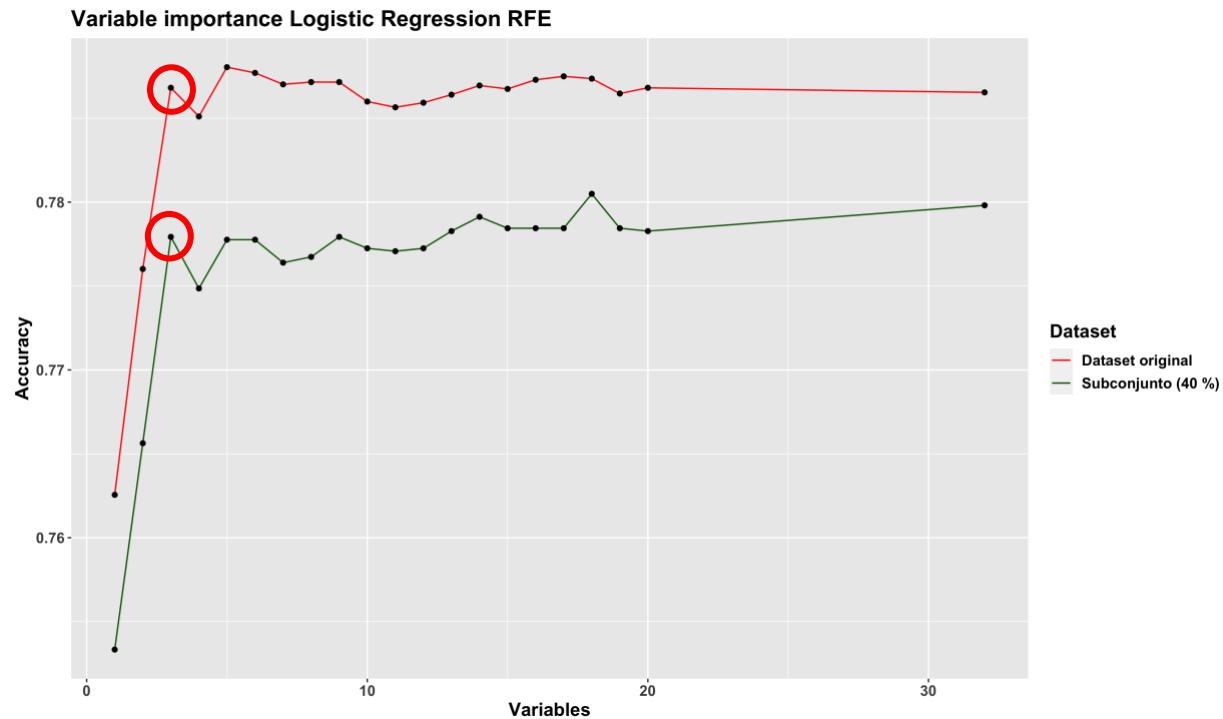


Figure 9. RFE Logistic Regression (hasta 20 variables)

Analizando el gráfico resultante, observamos que la diferencia de *Accuracy* en ambos *datasets* no es muy significativa: **mientras que con el dataset completo se alcanza un valor en torno a 0.78, con el subconjunto la precisión se reduce a tan solo 0.77**. De hecho, en ambos casos se obtienen muy buenos resultados con tan solo tres variables, y en ambos casos coinciden:

```

## Top 3 variables (coinciden con las tres primeras en stepwise AIC-BIC):

## [1] ccsMort30Rate
## [2] mortality_rsi
## [3] bmi

```

## 4.5 Recursive Feature Elimination (bajo Random Forest)

En el caso de *Random Forest*, **con 4-5 parámetros se obtienen resultados muy similares**: en el caso el *dataset* original en torno a 0.90 y 0.88 con tan solo el 40 % de las observaciones. De hecho, en el punto de máximo *accuracy* (con tan solo 4 variables en el *dataset* original y 5 en el subconjunto), ambos modelos comparten las mismas variables, a excepción de *ahrq\_ccs*:

```

##   n variable_dataset_original variable_subset
## 1 1             Age           Age
## 2 2      ccsMort30Rate   ccsMort30Rate
## 3 3    mortality_rsi  mortality_rsi
## 4 4            bmi         bmi
## 5 5             -        ahrq_ccs

```

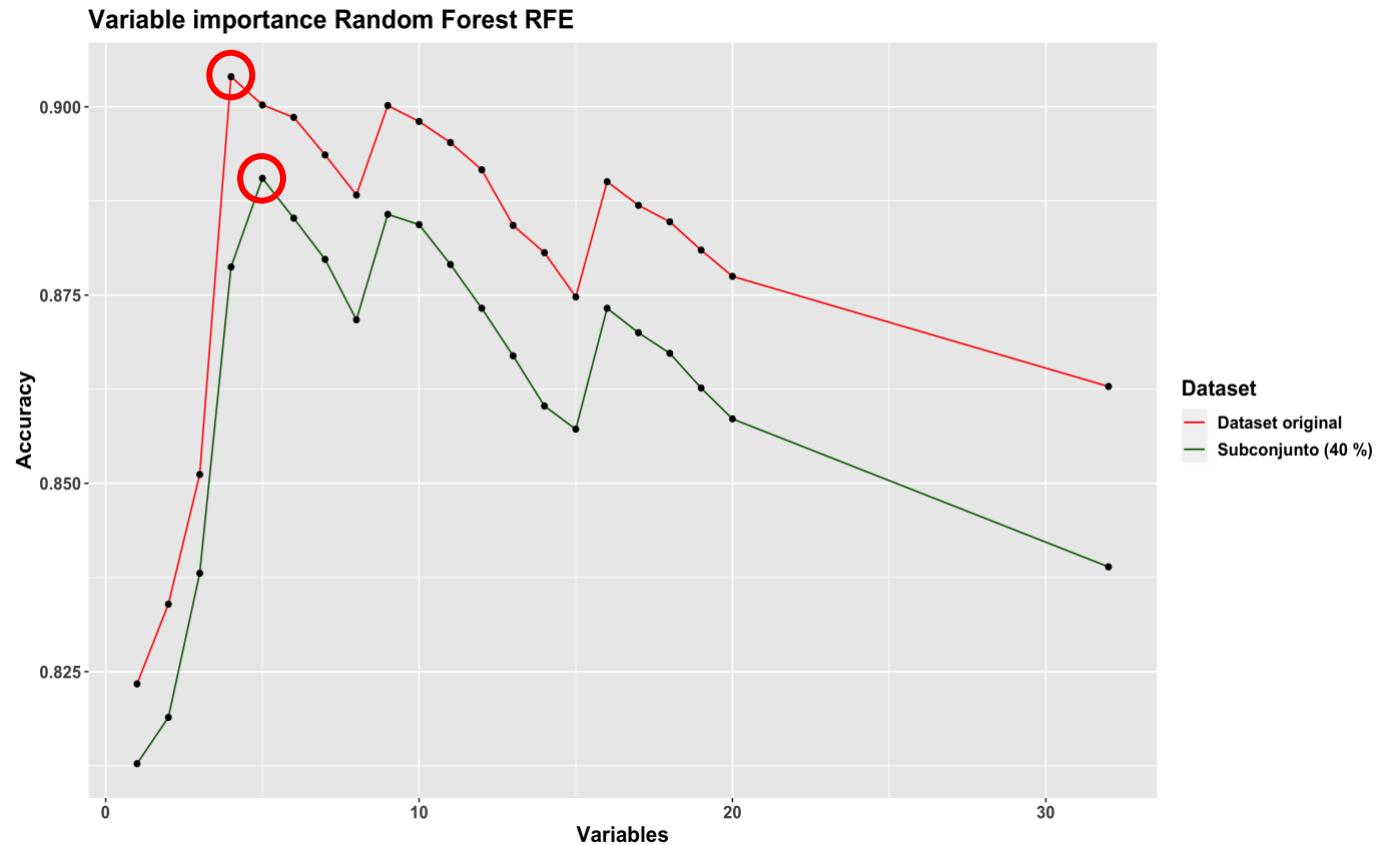


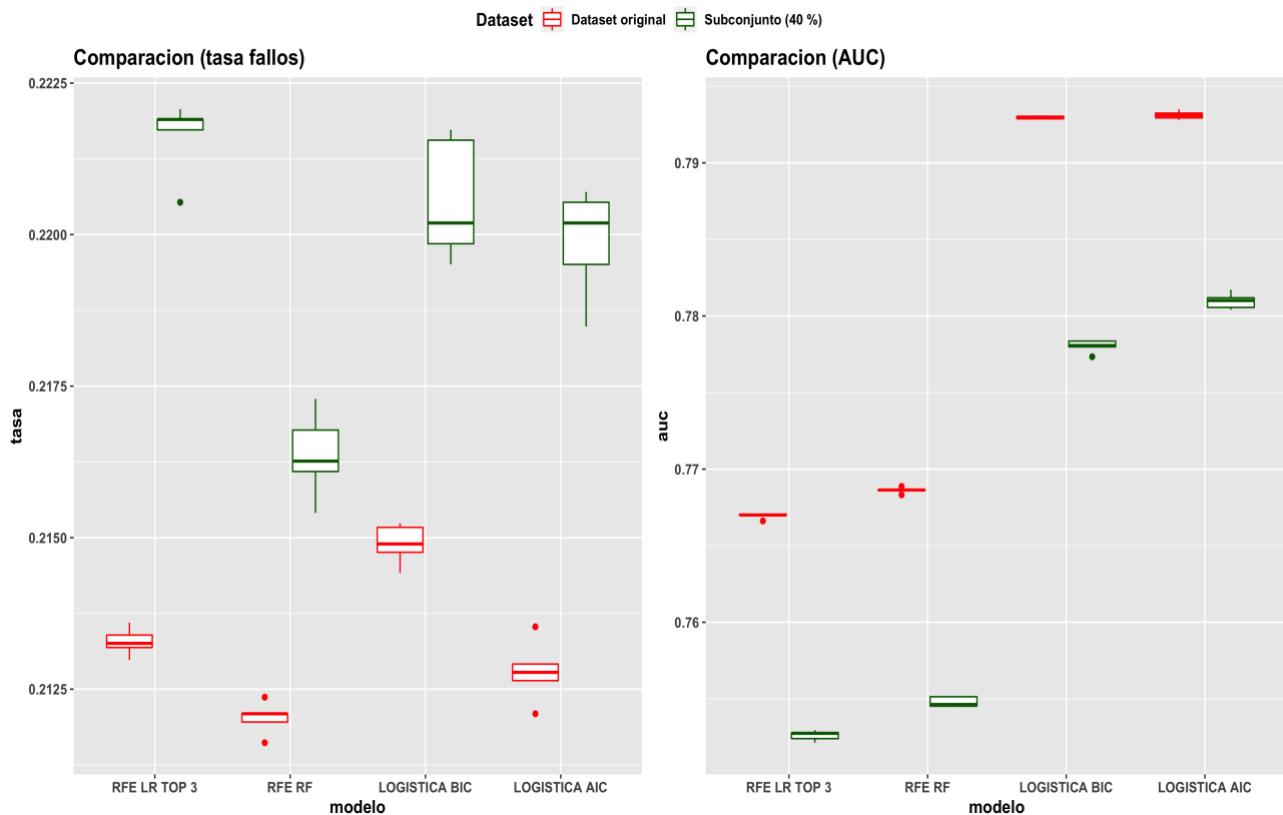
Figure 10. RFE Random Forest (hasta 20 variables)

Curiosamente, llama la atención la notable mejoría que experimenta al emplear un modelo no lineal como es el caso de *Random Forest*, **de nuevo un indicio de la no linealidad de las variables**.

## 4.6 Selección bajo logística

A continuación, y una vez realizada la selección de variables tanto por AIC/BIC como por RFE, **realizamos una primera comparación, bajo logística, de la selección de variables obtenidas en los métodos anteriores:**

1. Variables obtenidas por *stepwise AIC*: 17 en el *dataset* original y 14 en el subconjunto.
2. Variables obtenidas por *stepwise BIC*: 10 en el *dataset* original y 9 en el subconjunto.
3. Variables obtenidas por *Recursive Feature Elimination* (logística), concretamente el *top 3*.
4. Variables obtenidas por *Recursive Feature Elimination (Random Forest)*\*<sup>1</sup>, concretamente el *top 4* y *top 5* variables obtenidos con el *dataset* original y con el subconjunto, respectivamente.



*Figure 11. Comparación bajo logística (I)*

Como primera impresión, y dada la escala del eje Y, **la diferencia en cuanto a tasa de fallos y AUC se refiere es muy pequeña**, concretamente de 0.01, aproximadamente (0.21 en tasa de fallos y 0.77-0.79 en AUC frente a 0.22 y 0.76-0.78). Es decir, habiendo eliminado el 60 % de las observaciones del fichero original, la ganancia de error es muy pequeña, además de que la varianza en las "cajas" se comporta de forma similar en ambos casos.

Por tanto, dado que la diferencia entre ambos *datasets* es pequeña, **de cara al resto de la práctica se trabajó con el subconjunto del 40 %**. No obstante, en los últimos apartados (y

una vez tuneados los modelos), se realiza una última comparación con el fichero original, comprobando de este modo si el orden de los algoritmos se conserva.

#### 4.6.2 Selección de los mejores sets de variables

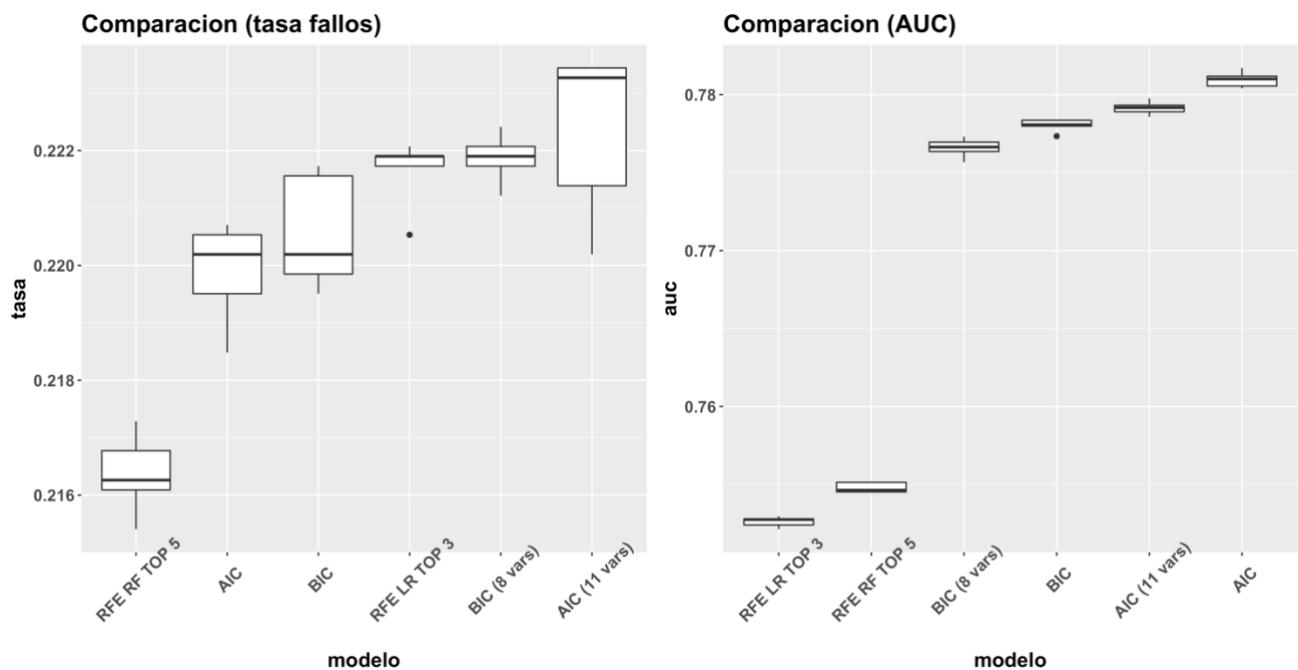
Una vez realizada la comparación con el *dataset* original, nos centramos en la selección de variables obtenida por el subconjunto, tanto por *stepwise AIC/BIC* como por RFE. En primer lugar, y remontándonos al diagrama de cajas anterior, debemos recordar la diferencia en el número de variables entre *stepwise AIC* y *BIC*: con 14 y 9 variables, respectivamente, **la tasa de fallos es prácticamente idéntica, y en relación con el valor AUC, la diferencia es de tan solo unas milésimas, pues ambos modelos se sitúan en torno a 0.78**.

Por tanto, el hecho de incluir demasiadas variables **no afecta en gran medida al modelo, lo cual puede traducirse en un sobreajuste en el resto de los algoritmos**. En consecuencia, en ambas selecciones **probamos a eliminar las variables categóricas menos relevantes, concretamente aquellas con un menor poder predictivo**, tal y como pudimos observar en el apartado de depuración, gracias al **Valor de Información (IV)**:

1. En el caso de *stepwise AIC*, eliminamos los campos *baseline\_cvd*, *asa\_status.0* y *baseline\_diabetes* (iv: 0.04, 0.0062 y 0.0013, respectivamente).
2. En el caso de *stepwise BIC*, eliminamos el campo *asa\_status.0* (iv: 0.0062).

```
##      n      stepwise_aic      stepwise_bic
## 1    1      mortality_rsi      mortality_rsi
## 2    2      ccsMort30Rate      ccsMort30Rate
## 3    3          bmi          bmi
## 4    4      month.8      month.8
## 5    5          dow.0          dow.0
## 6    6          Age          Age
## 7    7      moonphase.0      moonphase.0
## 8    8      month.0 baseline_osteoart
## 9    9 baseline_osteoart      -
## 10   10 baseline_charlson      -
## 11   11      ahrq_ccs      -
```

Analicemos tanto la tasa de fallos como el valor AUC:

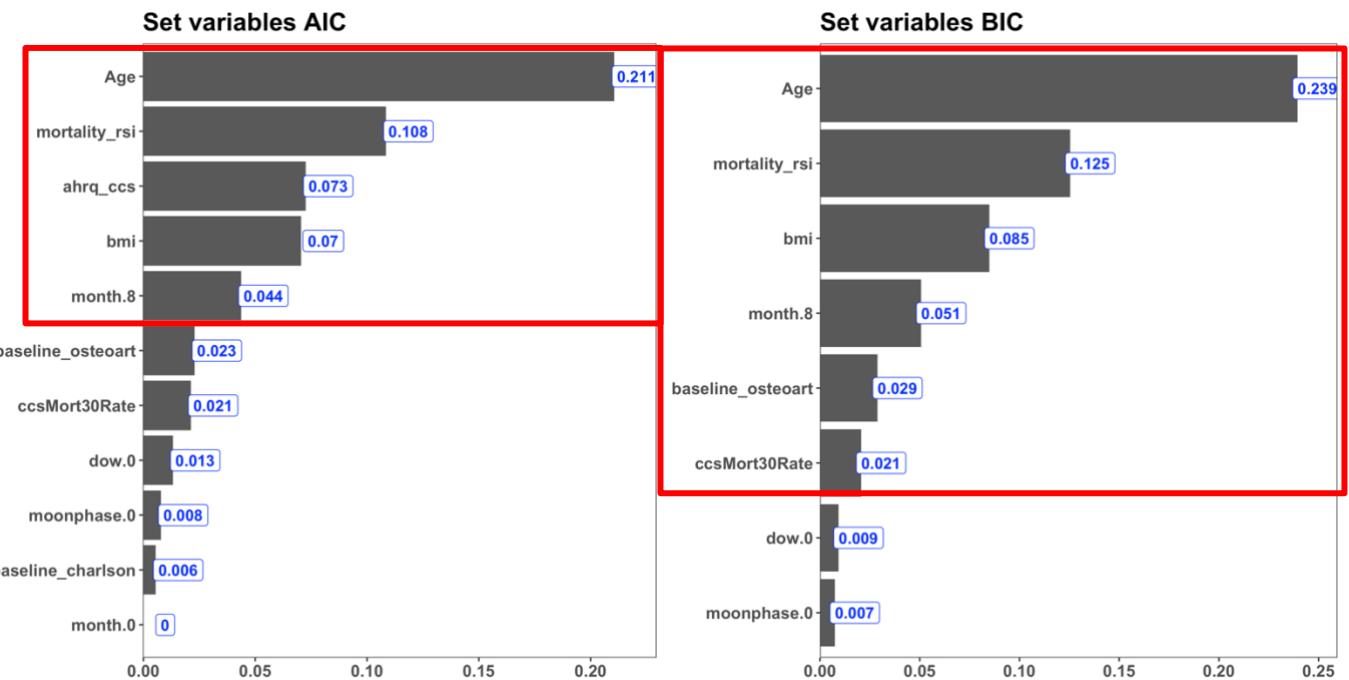


*Figure 12. Comparación bajo logística (II)*

Incluso reduciendo el número de variables en ambos casos, aunque la tasa de fallos aumente o el valor AUC disminuya ligeramente, la diferencia no es muy significativa. Además, aunque la varianza con el set de *stepwise AIC* (con 11 variables) parezca aumentar, lo cierto es que la variación no es tan significativa por la escala de los ejes (oscila entre 0.220 y 0.224-0.225, aproximadamente).

No obstante, uno de los aspectos que más llama la atención es la selección de variables RFE con *Random Forest*, **donde con tan solo 5 variables el *accuracy* aumentaba hasta cerca del 90 %**, mucho más alto que un modelo logístico. Por tanto, dado que con un modelo de árbol hemos conseguido obtener mejores resultados (lo que puede llegar a evidenciar la separación no lineal de los datos) ¿Y si entrenamos un pequeño modelo *Random Forest* para observar la importancia de las variables, tanto de la selección *stepwise AIC* como *BIC*? De este modo (y dado que con un modelo de árbol se obtienen mejor *accuracy*), podemos comprobar su relevancia en el modelo y, en caso necesario, descartar variables (a través de *MeanDecreaseAccuracy*):

```
#-- mtry: probamos a sortear 5 variables y 2000 arboles
rf_modelo_bic <- train_rf_model(surgical_dataset, formula.candidato.bic.2, ntree = 2000, mtry = 5, grupos = 5, repe = 5, nodesize = 10, seed = 1234)
```



*Figure 13. Importancia variables AIC - BIC (bajo Random Forest)*

Analizando el gráfico de importancia, caben destacar cuatro principales variables, **las cuales coinciden en ambos modelos, tanto en stepwise AIC como BIC**: *Age*, *mortality\_rsi*, *bmi* y *month.8*. Por otro lado, en el modelo *AIC* cabe destacar, además, la variable *ahrq\_ccs*. En relación con el modelo *BIC*, cabe destacar una quinta variable: *baseline\_osteoart*, aunque también la variable *ccsMort30Rate* se aproxima bastante. El resto de las variables, por el contrario, **no parecen afectar al modelo en caso de ser descartadas**.

Por tanto, vista la importancia que presentan las variables en el modelo *Random Forest*, ¿realmente es necesario un modelo con 11 u 8 variables, como es el caso de *stepwise AIC* o *BIC*? Es decir, **¿Y si reducimos el modelo a las 4 variables más relevantes?** Por ejemplo, una última comparación bajo logística consistiría en (y en base al gráfico de importancia anterior) analizar un modelo con las cuatro variables más importantes bajo *Random Forest*, comunes a ambos *stepwise*:

### 1. *Age, mortality\_rsi, bmi* y *month.8*

También podríamos probar las cuatro variables más importantes en *stepwise AIC*, que tan solo se diferencia por *ahrq\_ccs* **¿Influirá en mayor medida el tipo de intervención quirúrgica o si la intervención se realizó en el mes de septiembre?**

### 2. *Age, mortality\_rsi, bmi* y *ahrq\_ccs*

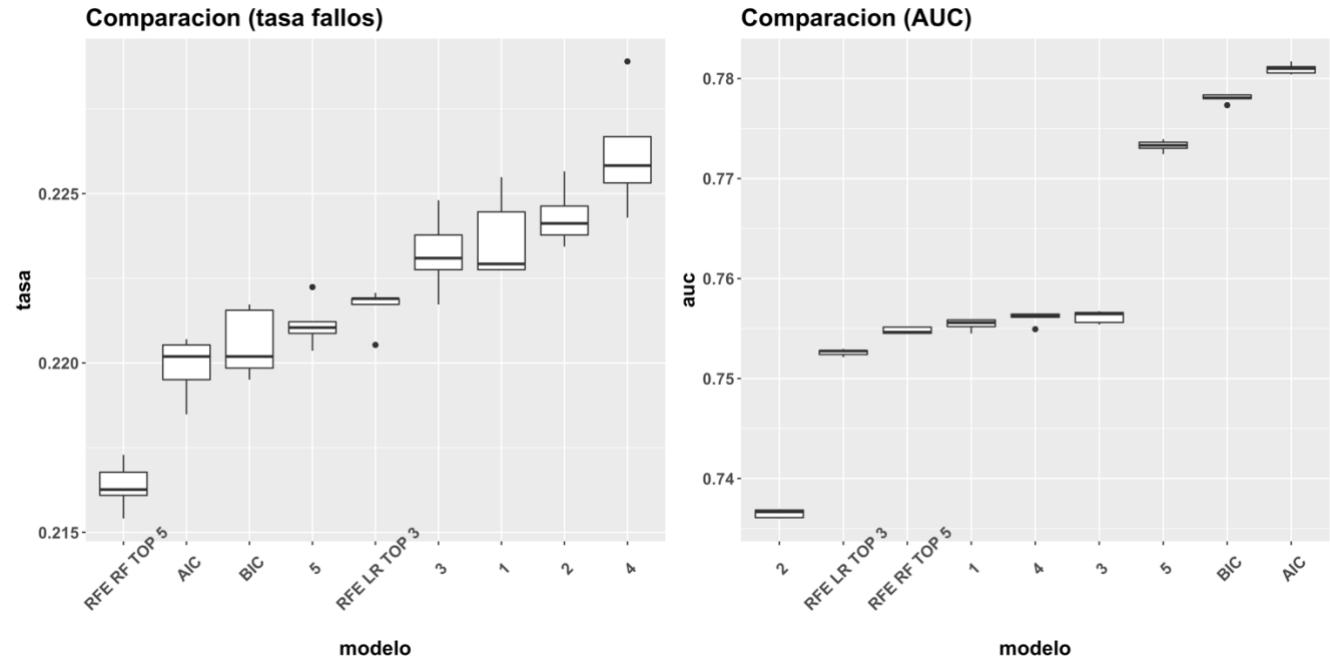
No obstante, ¿Podríamos tener en cuenta alguna variable adicional en los *sets* candidatos? Por ejemplo, del primer *set AIC* podríamos considerar ***month.8***, dado que su importancia en el modelo *Random Forest* se aproxima al de otras variables como *ahrq\_ccs* y *bmi*:

### 3. *Age, mortality\_rsi, bmi, ahrq\_ccs* y *month.8*

Por otro lado, en relación con el *set BIC*, variables como *baseline\_osteoart* y *ccsMort30Rate* son muy similares en cuanto a *Mean Decrease Accuracy* se refiere, por lo que también realizamos una prueba con ambas variables, junto con las cuatro más importantes:

4. *Age, mortality\_rsi, bmi, month.8 y baseline\_osteoart.*

5. *Age, mortality\_rsi, bmi, month.8 y ccsMort30Rate.*



**Figure 14. Comparación bajo logística (III)**

Analicemos los resultados obtenidos:

- **Modelos con 5 variables.** Si recordamos, la diferencia entre los *sets* con cinco variables es de tan solo una o dos *features*:

<i>set 3.</i>	<i>Age</i>	<i>mortality_rsi</i>	<i>month.8</i>	<i>bmi</i>	<i>ahrq_ccs</i>
<i>set 4.</i>	<i>Age</i>	<i>mortality_rsi</i>	<i>month.8</i>	<i>bmi</i>	<i>baseline_osteoart</i>
<i>set 5.</i>	<i>Age</i>	<i>mortality_rsi</i>	<i>ccsMort30Rate</i>	<i>bmi</i>	<i>month.8</i>
<i>RFE-RF TOP 5</i>	<i>Age</i>	<i>mortality_rsi</i>	<i>ccsMort30Rate</i>	<i>bmi</i>	<i>ahrq_ccs</i>

Es decir, todos ellos coinciden en que *Age*, *mortality\_rsi* y *bmi* son variables óptimas. Sin embargo, algunos de los *sets* escogen *mortality\_rsi*, otros *month.8* o *ahrq\_ccs*. ¿Cuáles son los adecuados? La respuesta reside en el gráfico anterior: **empleando el set 5, es decir, las variables *ccsMort30Rate* y *month.8***, el modelo logístico mejora en AUC de 0.75 en el resto de *sets* a 0.77, en la misma línea que las selecciones *stepwise AIC* y *BIC* originales; además de

reducir muy ligeramente la tasa de fallos. **Por tanto, de los sets candidatos con 5 variables, dado el mayor AUC que supone, nos decantamos por el set 5.**

- **Modelos con 3-4 variables.** Sobre estos últimos, debemos recalcar tres sets:

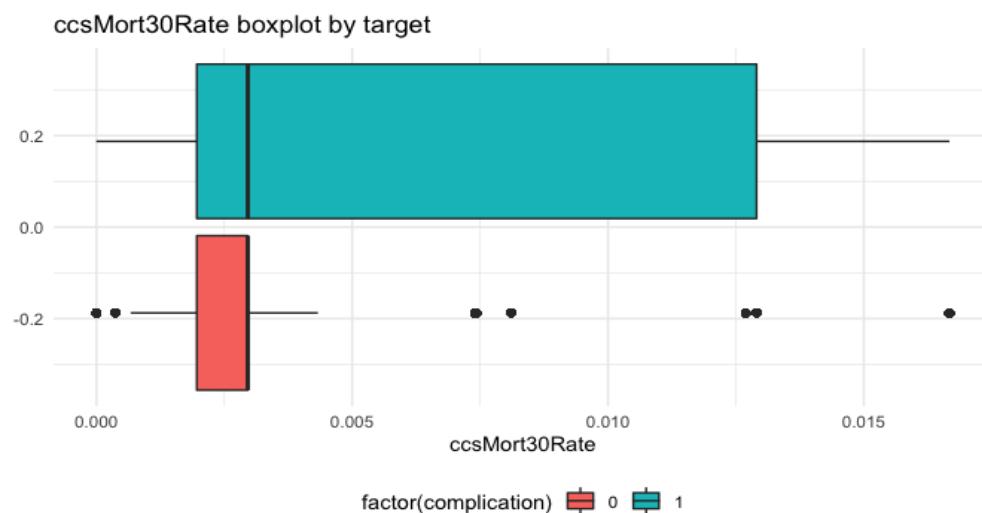
<i>set 1.</i>	<i>Age</i>	<i>mortality_rsi</i>	<i>bmi</i>	<i>month.8</i>
<i>set 2.</i>	<i>Age</i>	<i>mortality_rsi</i>	<i>bmi</i>	<i>ahrq_ccs</i>
<i>RFE-LR TOP 3</i>	<i>ccsMort30Rate</i>	<i>mortality_rsi</i>	<i>bmi</i>	-

Sobre estos candidatos, debemos destacar tanto el *set 2* como el top 3 obtenido con RFE sobre logística (dado que el *set 1* obtiene un peor AUC, por debajo de 0.74). Por tanto, ¿Cuál de los sets debemos escoger? En este caso, **se ha tomado la decisión de elegir el primero: *Age*, *mortality\_rsi*, *bmi* y *month.8***, principalmente por un motivo: al realizar el modelo de *Random Forest*, tanto por *RFE* como por la librería propia de *rf*, los resultados obtenidos han sido mucho mejores en comparación a un modelo logístico. Además, sobre el gráfico de importancia obtenido anteriormente, **se prima una mayor importancia a variables como *Age* o *month.8*, a diferencia de *ccsMort30Rate*.**

Dado que aparentemente los modelos de árbol obtienen mejores resultados, puede resultar más preferible escoger un *set* que, pese a no obtener muy buenos resultados en logística, si puede mejorar en *Random Forest*. Por tanto, **nos decantamos también por el set 1.**

```
##-- Modelo 1
var_modelo1 <- c("mortality_rsi", "ccsMort30Rate", "bmi", "month.8", "Age")
##-- Modelo 2
var_modelo2 <- c("mortality_rsi", "bmi", "month.8", "Age")
```

No obstante, antes de continuar, observamos que la diferencia entre ambos sets candidatos es de tan solo una variable: ***ccsMort30Rate*** ¿Qué distribución presenta con la variable objetivo?



*Figure 15. Distribución de *ccsMort30Rate* sobre la variable objetivo*

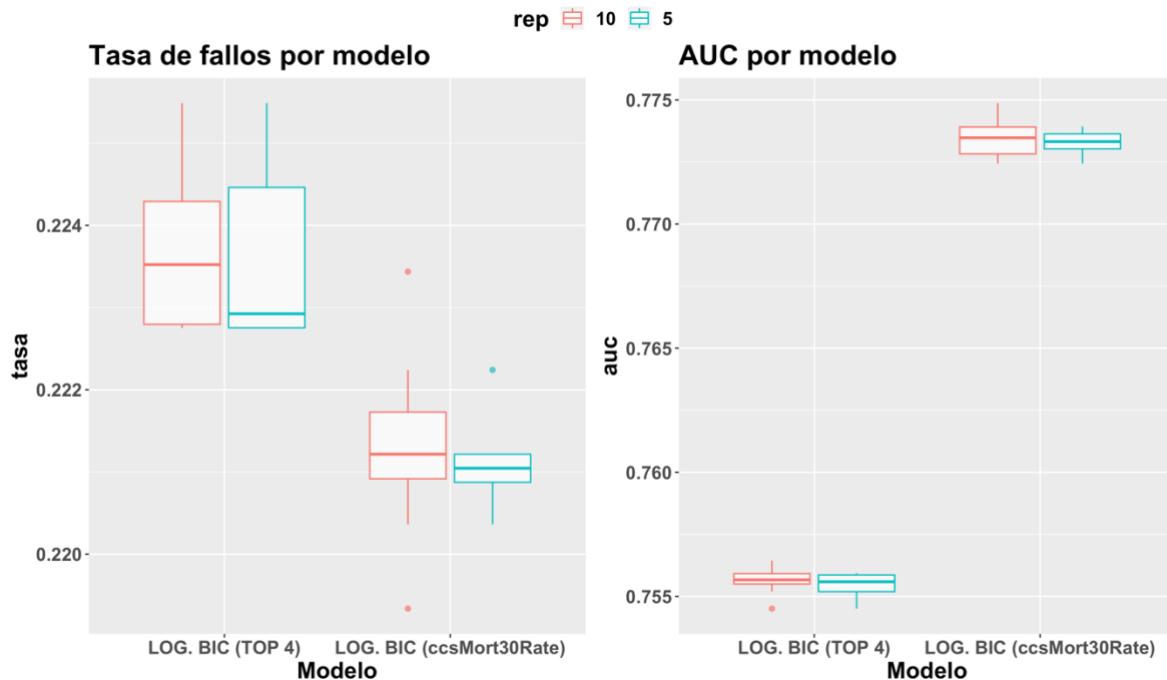
En este caso, con la variable *ccsMort30Rate* se aprecia una mejor separación lineal que en otras variables analizadas hasta el momento ¿Qué puede suponer? Aunque la separación no es prácticamente perfecta, pues ambas clases se solapan en algunos valores (por debajo de 0.005), es muy probable que, en caso de utilizar modelos no lineales como es el caso de *Random Forest* o *gbm*, la diferencia entre ambos *sets* (de cuatro y cinco variables) no sea muy relevante o se obtenga prácticamente el mismo *accuracy*.

Por el contrario, utilizando clasificadores lineales como es el caso de la regresión logística, el modelo mejora ligeramente (gracias al uso de *ccsMort30Rate*), por lo que es posible que con otros modelos lineales como la *svm* con *kernel* lineal, el modelo ganador será empleando el *set* de cinco variables.

No obstante, el hecho de que un *set* de variables sea mejor o peor en un modelo lineal o no lineal es algo que podemos intuir, pero **a priori** lo que necesitamos para demostrarlo son resultados empíricos. Por tanto, de cara a la elaboración de los modelos obligatorios empleamos ambos *sets* candidatos.

#### 4.7 Tuneo y comparación final

Finalmente, realizamos una última comparación de ambos modelos candidatos, **aumentando la validación cruzada de 5 a 10 repeticiones**:



*Figure 16. Comparación final logística (5-10 rep.)*

A primera vista, al aumentar el número de repeticiones, la varianza en ambos modelos es muy similar, con una ventaja en el segundo modelo en cuanto a AUC se refiere (0.77 frente a 0.75, aproximadamente). Además, aunque la varianza con tan solo 4 variables aparente ser mayor (en la tasa de fallos), la escala del eje puede conducir a engaño, ya que la variación es del orden de milésimas.

**Nota:** a partir de ahora, y de cara a la elaboración de los modelos, denotaremos ambos *sets* como:

- **Modelo 1:** modelo elaborado con el primer *set* de variables (5 parámetros).
- **Modelo 2:** modelo elaborado con el segundo *set* de variables (4 parámetros).

## 5. Modelos iniciales con H2O

Una vez realizada la selección de variables y decantarnos por dos posibles *sets*, antes de comenzar con el tuneo de hiperparámetros **realizamos un modelo *autoML* por cada *set* de variables**. De este modo podremos comprobar, de forma orientativa:

1. Qué modelo o modelos son los más adecuados, tanto a nivel AUC como en tasa de fallos.
2. Del mejor modelo, qué parámetros se ha empleado.

### 5.1 Modelo 1

```
#  Modelo 1 (extraemos los 10 mejores modelos segun h2o)
aml_1 <- h2o.automl(x = var_modelo1, y = target, max_models = 10,
                      training_frame = surgical_dataset_h, nfolds = 5, seed = 1234)

      model_id.      auc    logloss  mpc_error
1        GBM_5_AutoML 0.9211350 0.2591059 0.1608837
2        StackedEnsemble 0.9208431 0.2647496 0.1603029
3          GBM_1_AutoML 0.9200581 0.2690935 0.1659645
4        StackedEnsemble 0.9200033 0.2647962 0.1676764
5          GBM_2_AutoML 0.9190857 0.2695043 0.1648937
6        XGBoost_3_AutoML 0.9189663 0.2613378 0.1651901
7      XGBoost_grid_1_AutoML 0.9189402 0.2629065 0.1675371
8        GBM_grid_1_AutoML 0.9187735 0.2763313 0.1689585
9          GBM_3_AutoML 0.9175951 0.2725404 0.1757392
10       XGBoost_2_AutoML 0.9169047 0.2717032 0.1707306

*mpc_error: mean_per_class_error
```

En un primer análisis, de todos los modelos *autoML* creados, **modelos no lineales como *gradient boosting*, *ensemble* y *xgboost* obtienen los mejores resultados**, con un AUC en torno a 0.91-0.92, así como un error medio por cada clase de 0.16-0.17, aproximadamente. Por otro lado, si analizamos los parámetros del mejor modelo, concretamente *gradient boosting*:

```
##           modelo ntrees max_depth sample_rate col_sample_rate
## 1 Modelo 1 (BIC TOP 5)     82        15         0.8            0.8
##   col_sample_rate_per_tree
## 1                  0.8
```

Analizando los parámetros, por lo general *h2o* opta por un modelo *gbm* sencillo, con un número bajo de árboles (inferior a 100), una profundidad moderada en cada árbol (15), además de sortear no solo las observaciones, sino además variables (concretamente el 80 %).

## 5.2 Modelo 2

	model_id	auc	logloss	mpc_error
1	GBM_5_AutoML	0.9096419	0.2700148	0.1646700
2	StackedEnsemble_BestOfFamily_AutoML	0.9088350	0.2712222	0.1666722
3	StackedEnsemble_AllModels_AutoML	0.9083655	0.2708531	0.1683963
4	XGBoost_3_AutoML	0.9083145	0.2683037	0.1735557
5	GBM_1_AutoML	0.9075638	0.2769396	0.1730899
6	GBM_2_AutoML	0.9072236	0.2767946	0.1770094
7	XGBoost_grid_1	0.9065834	0.2696259	0.1678335
8	XGBoost_2_AutoML	0.9064648	0.2756899	0.1671682
9	XGBoost_1_AutoML	0.9047755	0.2746863	0.1725274
10	GBM_grid_1_AutoML	0.9044822	0.3115999	0.1720491

\*mpc\_error: mean\_per\_class\_error

Nuevamente, con el segundo *set* de variables **continúa optando por un modelo *gradient boosting*, con un AUC muy similar al primer set de variables** (0.90 frente a 0.92). Dado que los modelos logísticos obtenidos hasta el momento apenas superan un valor de 0.7 en AUC, el hecho de obtener un mejor resultado con *gradient boosting* resulta ser, nuevamente, un **indicio de la no linealidad de los datos**.

Además, si analizamos los parámetros del mejor modelo:

```
##                               modelo ntrees max_depth sample_rate col_sample_rate
## 1    Modelo 1 (BIC TOP 5)     82        15         0.8          0.8
## 2 Modelo 2 (AIC-BIC-top4)    72        15         0.8          0.8
##   col_sample_rate_per_tree
## 1                      0.8
## 2                      0.8
```

Salvo el número de árboles, el resto de los parámetros coinciden. Por tanto, de cara a la elaboración de los modelos **comprobamos si el mejor modelo en ambos sets es, efectivamente, un modelo *gbm*, o si al menos se sitúa entre los mejores modelos con un AUC y parámetros similares**.

## 6. Redes neuronales

### 6.1 Modelo 1

A continuación, procedemos con el tuneado del primer modelo *Machine Learning*: la red neuronal, comenzando con el primer *set* (5 variables *input*). En primera instancia, analicemos el número de nodos necesarios para obtener entre 20 y 30 observaciones por parámetro, teniendo 5 variables *input*:

1. Con 20 observaciones por parámetro:

$$h * (k + 1) + h + 1 = 5854/20 = 292 \text{ parámetros}$$

Con  $k = 5$  variables *input*, entonces:  $7 * h + 1 = 292$ . Es decir, 41 – 42 nodos

2. Con 30 observaciones por parámetro:

$$h * (k + 1) + h + 1 = 5854/30 = 195 \text{ parámetros}$$

Con  $k = 5$  variables input, entonces:  $7 * h + 1 = 195$ . Es decir, 27 – 28 nodos

En primera instancia, dado que dispone de tan solo cinco variables , probablemente no serán necesarios tantos nodos, a no ser que la complejidad del modelo suponga una gran mejoría. No obstante, realizamos un primer tuneo con varios tamaños (desde 5 hasta 40 nodos):

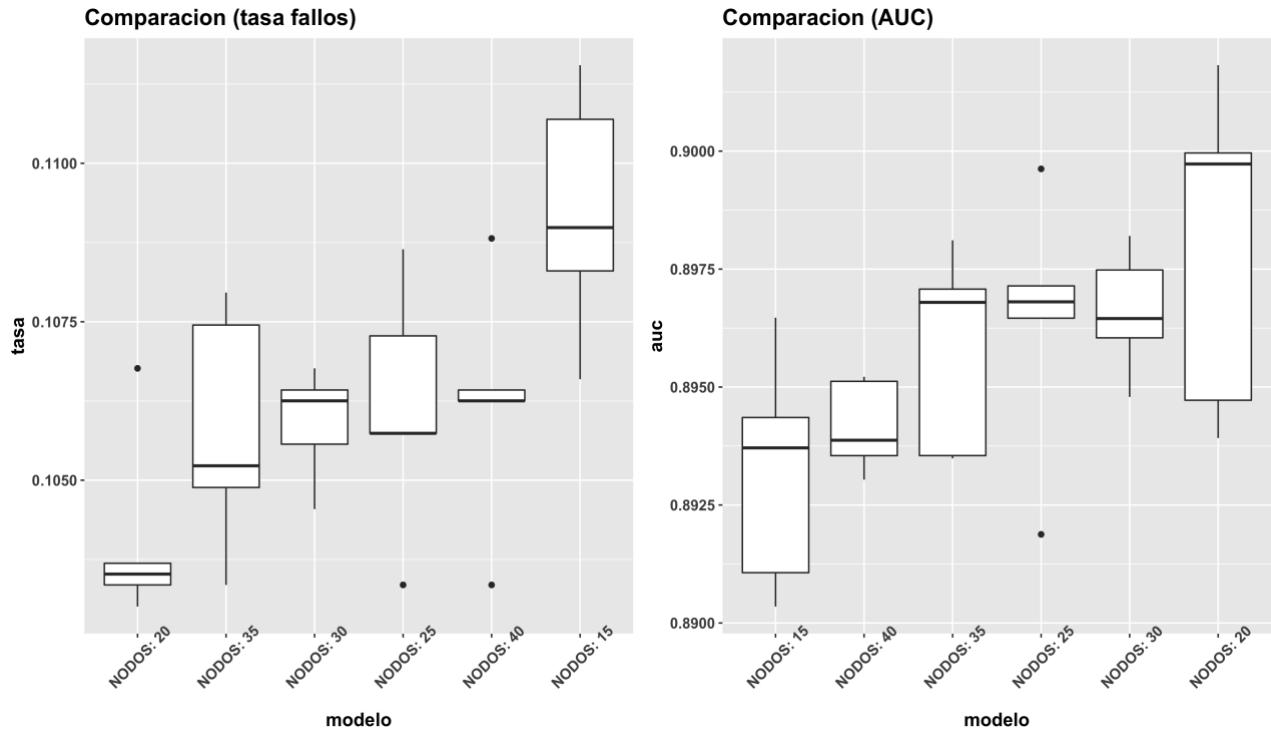
```
size.candidato.1 <- c(5, 10, 15, 20, 25, 30, 35, 40)
decay.candidato.1 <- c(0.1, 0.01, 0.001)

cvnnet.candidato.1 <- cruzadaavnnetbin(data=surgical_dataset,vardep=target,
                                         listconti=var_modelo1, listclass=c(""),
                                         grupos=5,sinicio=1234,repe=5, size=size.candidato.1,
                                         decay=decay.candidato.1,repeticiones=5,itera=200)

##   id size decay Accuracy
## 1  1   25 0.010 0.8950798
## 2  2   30 0.010 0.8947370
## 3  3   20 0.010 0.8945332
## 4  4   35 0.010 0.8939514
## 5  5   30 0.001 0.8934057
## 6  6   40 0.010 0.8921747
## 7  7   25 0.001 0.8919025
## 8  8   35 0.001 0.8914928
## 9  9   15 0.010 0.8905699
##          [...]
## 18 18  15 0.001 0.8794649
## 19 19  10 0.010 0.8793648
## 20 20  10 0.100 0.8653218
## 21 21  10 0.001 0.8557885
## 22 22    5 0.010 0.8237056
## 23 23    5 0.100 0.8129454
## 24 24    5 0.001 0.8113744
```

Analizando la tabla anterior, empleando un *decay* o *learning rate* de 0.01, el modelo obtiene muy buenos resultados, mejores incluso que la regresión logística, un indicativo nuevamente de la no linealidad de las variables. Por otro lado, y en relación con el número de nodos ¿Merece aumentar hasta 20, 30 o incluso 40 nodos? En vista a los resultados, no lo parece: a modo de ejemplo, con tan solo 15 nodos y un *decay* de 0.01, el valor de *accuracy* alcanza un 89 %, mientras que con 20, 30 o 40 nodos, la diferencia es de tan solo unas milésimas (no estamos ganando lo suficiente como para decantarnos por un modelo más complejo).

Por tanto, una buena alternativa sería emplear 15 nodos (10 tampoco sería una mala opción, aunque comienza a decaer hasta el 87 %). No obstante, con un *decay* de 0.01, analicemos tanto el sesgo como la varianza en base al número de nodos:

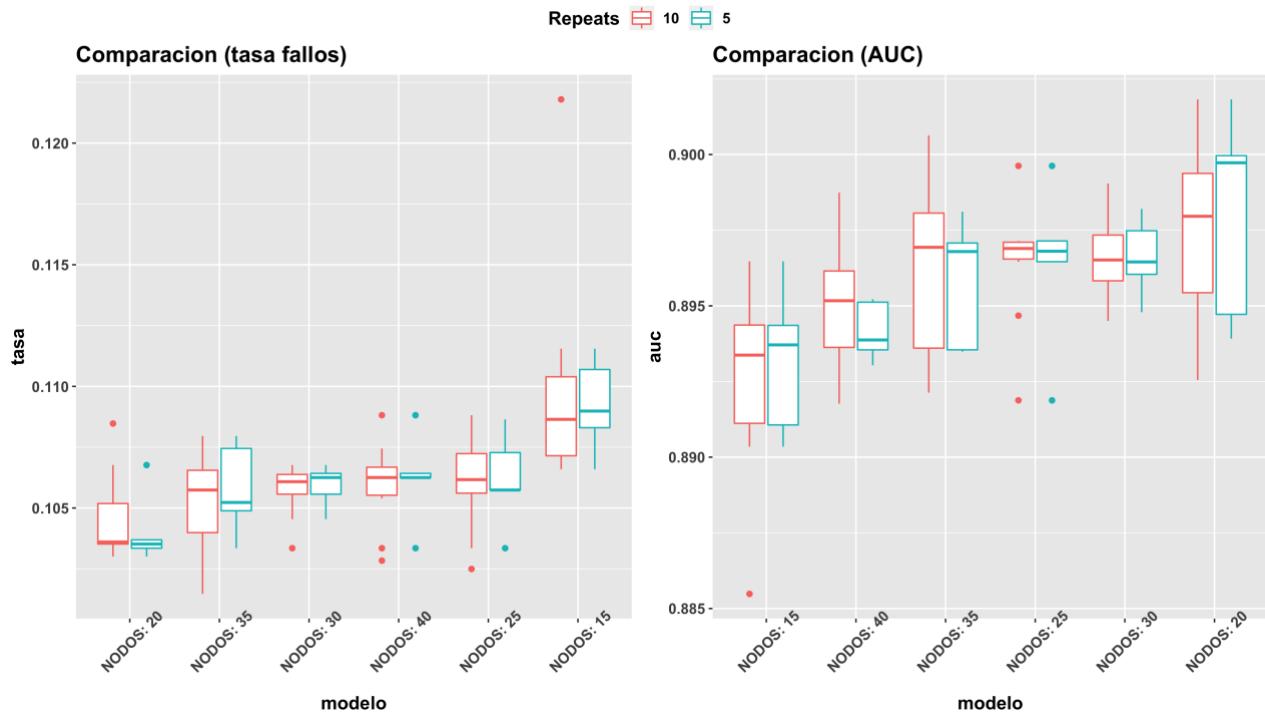


*Figure 17. Comparación avnnet modelo 1*

Como primera impresión, y en base al mejor AUC, un modelo con 20, 25 o 30 nodos sería una buena alternativa, de no ser por un aspecto clave: **la escala de los ejes**. En relación con la tasa de fallos, la diferencia de error entre un modelo con 15 nodos y un modelo con 20 es muy pequeña (de 0.11 a 0.105), además de que la varianza del modelo con 15 nodos (elevada desde un punto de vista gráfico), puede llevar a engaño, debido a la escala del eje: bien es cierto que la varianza observada en las "cajas" es meramente indicativo, por lo que en predicciones puntuales de una sola observación la variabilidad del error es mucho mayor. No obstante, la oscilación que presenta con tan solo 15 nodos es ínfima, entre 0.1075 y 0.11, mientras que aumentando la complejidad del modelo a 20 o 30 nodos, dicha oscilación se reduce en pocas centésimas (**varianza que incluso puede depender no solo del número de nodos, sino de la estructura de remuestreo o incluso el valor de la semilla**).

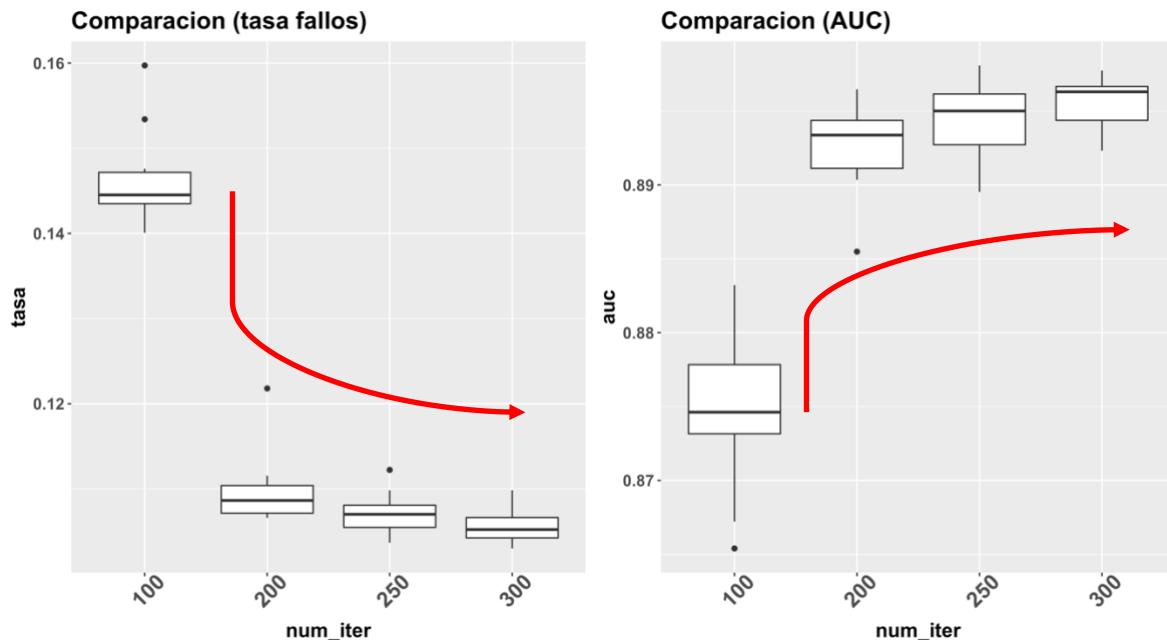
Por otro lado, la diferencia del AUC entre un modelo con 15 nodos y un modelo con 20 es muy pequeña (en torno 0.8925-0.895 en el caso de 15 nodos y 0.895-0.90 en el caso de 20 nodos).

De hecho, si aumentamos el número de repeticiones a 10:



*Figure 18. Comparación avnnet modelo 1 (10 rep.)*

Comprobamos que tanto el orden de los diferentes modelos como su varianza se mantienen prácticamente constantes. En conclusión, con el primer *set* de variables **nos decantamos por un modelo de red con 15 nodos**, dado que la ganancia que supone al aumentar el número de nodos es muy pequeña, lo cual puede conducir al sobreajuste. A continuación, y aumentando a 10 el número de repeticiones, **probamos a tunear el número de iteraciones**:



*Figure 19. Comparación avnnet modelo 1 (num. iteraciones)*

Analizando los resultados, **bien es cierto que conforme aumenta el número de iteraciones, tanto la tasa de fallos como el AUC comienzan a estabilizarse**, especialmente de 100 a 200. No obstante, la ganancia que supone aumentar el número de iteraciones (a partir de 200) es muy pequeña: a modo de ejemplo, de 200 a 300 iteraciones, la tasa de fallos mejora de 0.11 a 0.105, aproximadamente, mientras que el valor AUC apenas se ve afectado. Por tanto, **mantenemos el número de iteraciones a 200**.

## 6.2 Modelo 2

A continuación, realizamos los mismos pasos con el segundo *set* de variables candidato. En primer lugar, y dado que disponemos de 4 variables *input*, para obtener 20 o 30 observaciones por parámetros necesitamos:

1. Con 20 observaciones por parámetro:

**Con  $k = 4$  variables input, entonces:  $6 * h + 1 = 292$ . Es decir, 48 – 49 nodos**

2. Con 30 observaciones por parámetro:

**Con  $k = 4$  variables input, entonces:  $6 * h + 1 = 195$ . Es decir, 27 – 28 nodos**

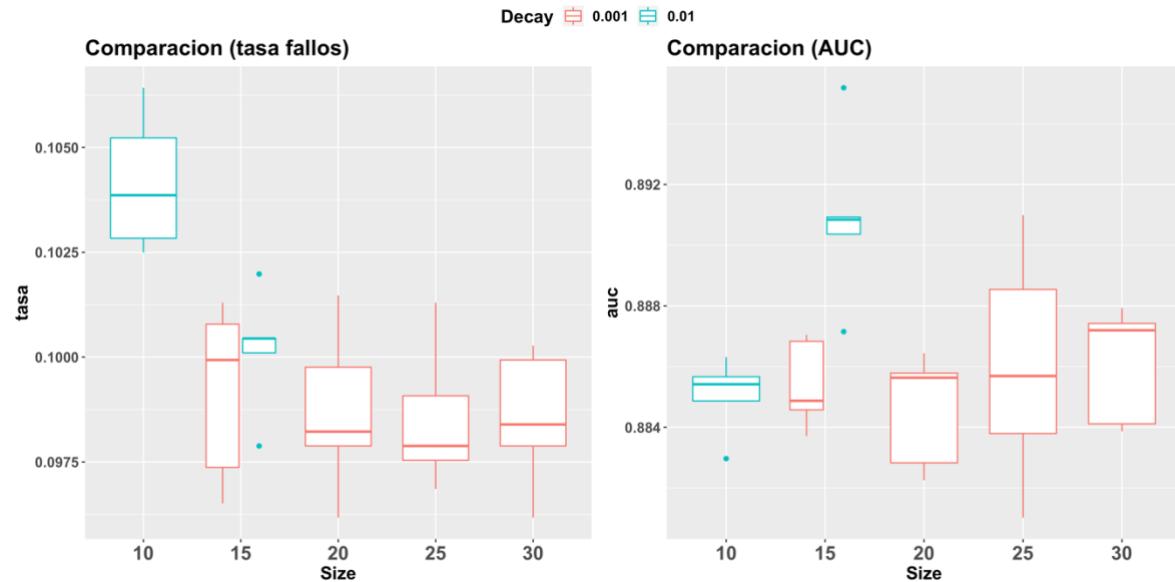
Con un primer cálculo, necesitaríamos un elevado número de nodos para obtener, al menos, 20 o 30 observaciones por parámetro. Sin embargo, si nos fijamos en lo empírico:

```
size.candidato.2 <- c(5, 10, 15, 20, 25, 30, 35, 40, 45)
decay.candidato.2 <- c(0.1, 0.01, 0.001)

cvnnet.candidato.1 <- cruzadaavnnetbin(data=surgical_dataset, vardep=target,
                                         listconti=var_modelo2, listclass=c(""),
                                         grupos=5, sinicio=1234, repe=5, size=size.candidato.1,
                                         decay=decay.candidato.1, repeticiones=5, itera=200)

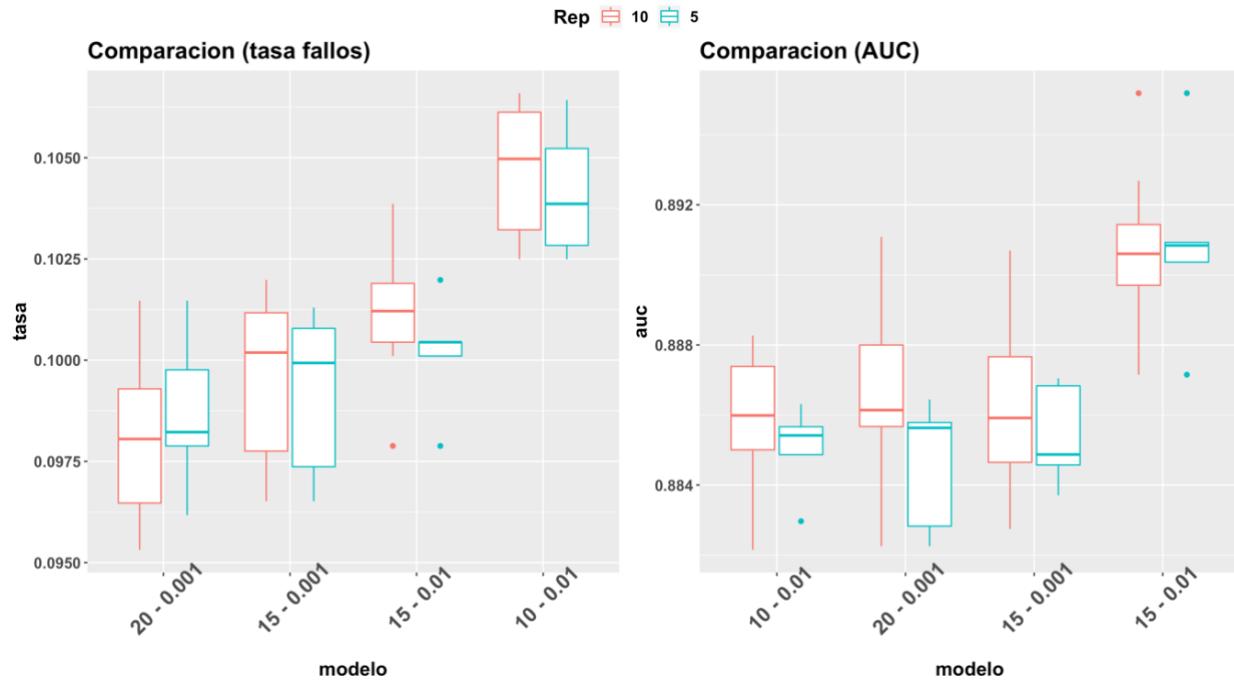
##   id size decay Accuracy
## 1  1  20 0.001 0.9026305
## 2  2  35 0.001 0.9021868
## 3  3  25 0.001 0.9020152
## 4  4  30 0.001 0.9017081
## 5  5  25 0.010 0.9007853
## 6  6  30 0.010 0.8998632
## 7  7  15 0.001 0.8997609
## 8  8  45 0.001 0.8995550
## 9  9  20 0.010 0.8994869
## 10 10  45 0.010 0.8994521
## 11 11  40 0.001 0.8994190
## 12 12  15 0.010 0.8992479
## 13 13  35 0.010 0.8989061
## 14 14  40 0.010 0.8986332
## 15 15  10 0.010 0.8946358
## 16 16  10 0.001 0.8906726
##           [...]
## 25 25  5 0.010 0.8447890
## 26 26  5 0.001 0.8358688
## 27 27  5 0.100 0.8303017
```

En un primer resultado, **con un parámetro de regularización pequeño (0.001) obtenemos buenos resultados**. A modo de ejemplo, llama la atención modelos de red con 20, 25 o 30 nodos que alcanzan un *Accuracy* de 0.90. Sin embargo, conforme descendemos en la tabla (y con ello, el número de nodos) comprobamos que la diferencia no es muy significativa: con tan solo 15 o 10 nodos y un *decay* de 0.01, el valor de *Accuracy* tan solo empeora apenas una centésima (0.90 a 0.89). Es decir, a simple vista el hecho de aumentar el número de nodos **no compensa la ganancia d**. Por tanto, analicemos tanto el sesgo como la varianza con los mejores modelos:



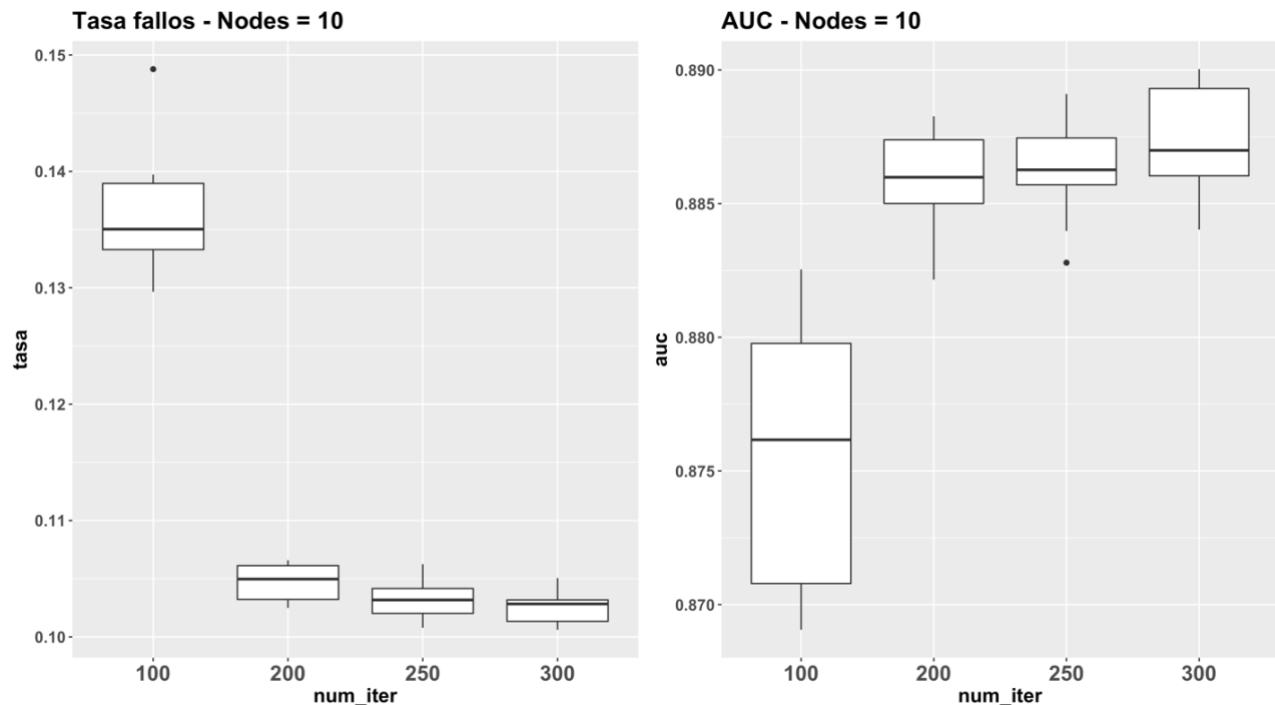
*Figure 20. Comparación avnnnet modelo 2*

En un primer resultado, **la ganancia de error que supone reducir el número de nodos a 10 (y con un decay de 0.01), es muy pequeña: de 0.09 con 25 nodos a 0.105 con tan solo 10 nodos**. Del mismo modo sucede con el área bajo la curva ROC, donde la mejoría con 15 o 30 nodos es de tan solo unas milésimas de diferencia, lo cual puede depender de condiciones azarosas como la semilla de aleatoriedad. A continuación, probamos a aumentar el número de repeticiones con un número de nodos bajo (10, 15 y 20):



*Figure 21. Comparación avnet modelo 2 (10 rep.).*

Incluso aumentando a 10 repeticiones, los resultados de los modelos no cambian apenas. Por tanto, dada su simplicidad y la poca ganancia de error que supone, para el segundo *set* de variables **elegimos un modelo con 10 nodos y decay 0.01**. Por último, en relación con el número de iteraciones:



*Figure 22. Comparación avnet modelo 2 (num. iteraciones)*

Del mismo modo que sucedía con el primer *set* de variables, **aumentar el número de iteraciones no supone una mejoría significativa, por lo que lo mantenemos a 200.**

### 6.3 Comparación final

Una vez obtenidos ambos modelos, realizamos la comparación de ambas redes junto con logística:

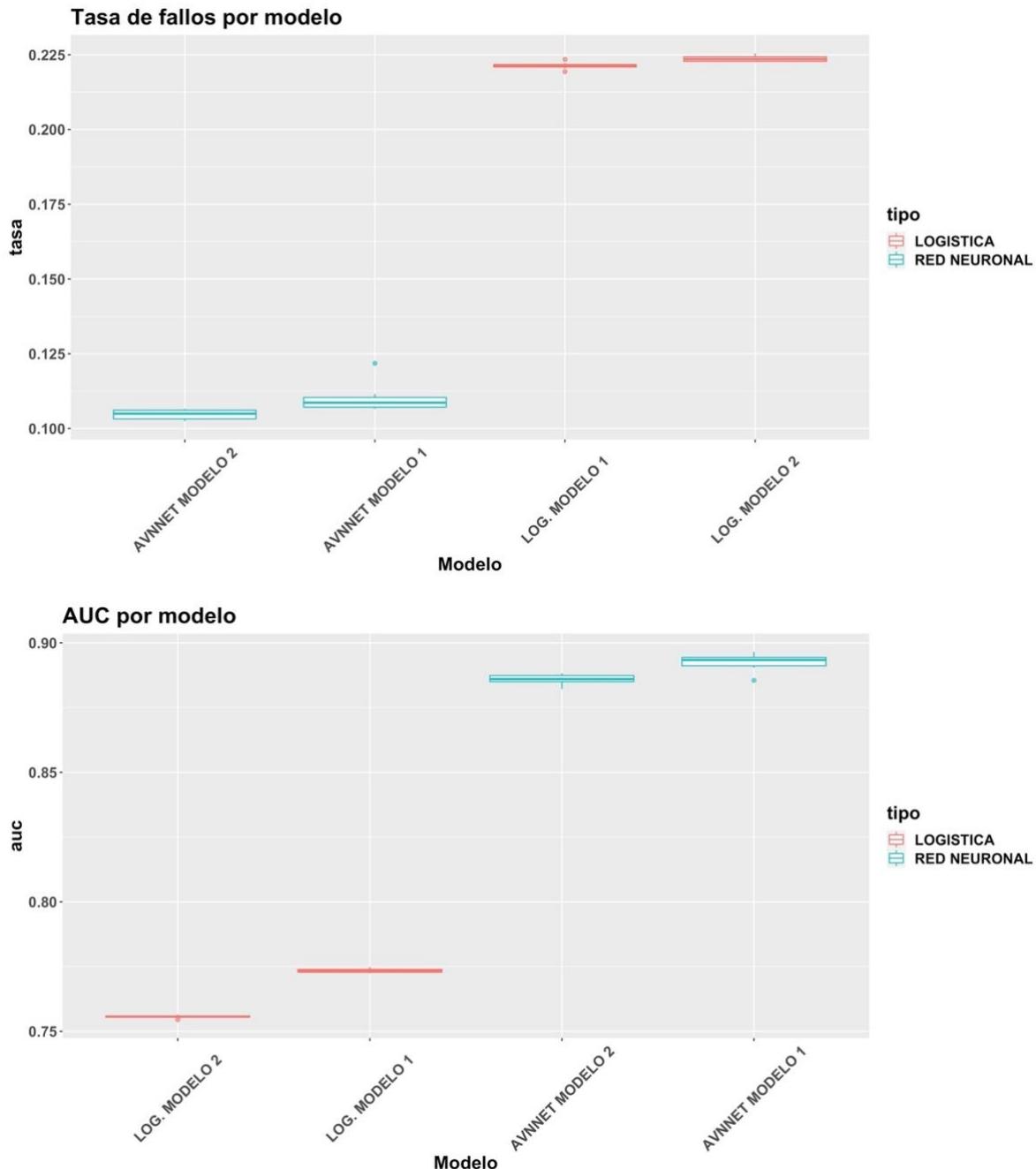


Figure 23. Comparación logística-avnnet

A la vista de los resultados obtenidos, tanto en tasa de fallos como en AUC, **los modelos de red mejoran significativamente los resultados del modelo**, un claro indicio (del mismo modo que en *h2o*) de la **no linealidad con la variable objetivo**. Por otro lado, en relación con ambos *sets* de variables, el hecho de añadir una variable *input* adicional (*ccsMort30Rate*) en el primer modelo **no hace mejorar significativamente sus resultados** ¿Mejorará ante modelos lineales como *svm* lineal?

### RESUMEN red neuronal:

1. Modelo 1: *size* = 15 y *decay* = 0.01.
2. Modelo 2: *size* = 10 y *decay* = 0.01.

## 7. Bagging

### 7.1 Selección del número de árboles

Continuando con el modelo *bagging*, de forma previa al tuneo de hiperparámetros, **debemos fijar un número de árboles (ntrees) para ambos sets de variables**, un valor mínimo a partir del cual el error OOB (*Out of bag error*) se estabiliza:

```
#-- Seleccion del numero de arboles
set.seed(1234)
## Modelo 1
rfbis.1<-randomForest(factor(target)~mortality_rsi+ccsMort30Rate+bmi+month.8+Age
                        data=surgical_dataset,
                        mtry=mtry.1,ntree=5000,nodesize=10,replace=TRUE)
## Modelo 2
rfbis.2<-randomForest(factor(target)~Age+mortality_rsi+bmi+month.8,
                        data=surgical_dataset,
                        mtry=mtry.2,ntree=5000,nodesize=10,replace=TRUE)
```

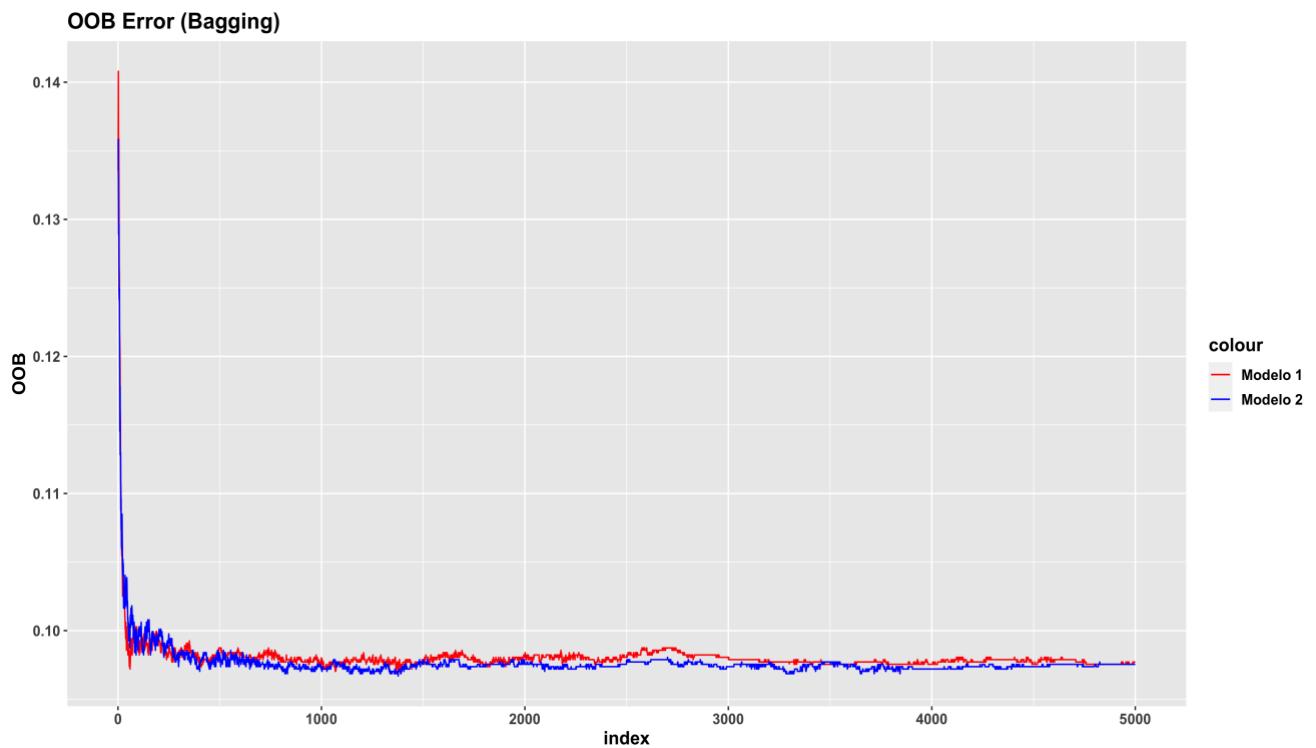


Figure 24. OOB Error (Modelos 1 y 2) (I)

Analizando el error *Out of bag*, en ambos modelos el error se estabiliza con aproximadamente 900-1000 árboles. Si hacemos zoom sobre el gráfico:

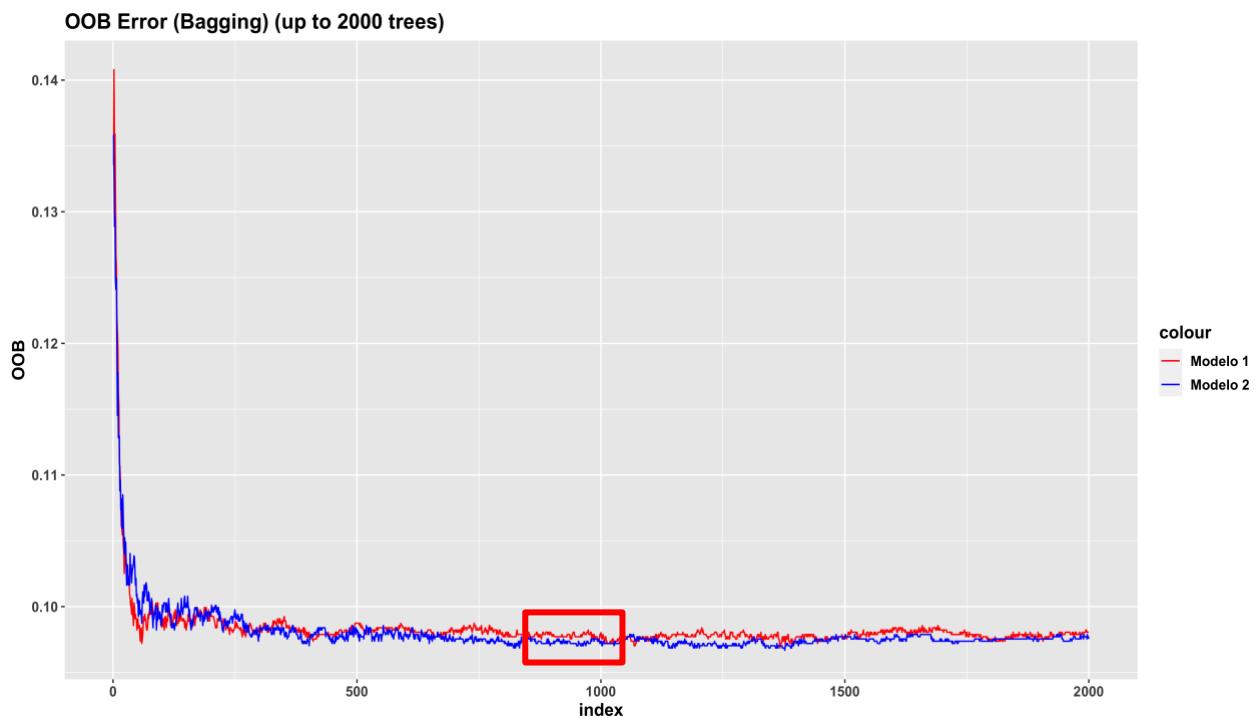


Figure 25. OOB Error (Modelos 1 y 2) (II)

Observamos como a partir de 200-250 árboles, el error comienza a situarse por debajo de 0.10. Sin embargo, **no es hasta los 900-1000 árboles cuando el error prácticamente se estabiliza, a partir del cual se detectan ciertas fluctuaciones (subidas o bajadas en el error), aunque de forma aleatoria**. Por tanto, **para ambos modelos escogemos 900 como número de árboles**. Aunque la ganancia en el error no sea muy significativa, simplemente estabilizamos la tasa de fallos:

```
n.trees.1 <- 900; n.trees.2 <- 900
```

## 7.2 Modelo 1

En un primer comienzo, hagamos un repaso previo de los parámetros a tunear en un modelo *bagging*, junto con el número de árboles:

- *mtry*: número de variables sorteadas aleatoriamente en cada división del árbol. Dado que se trata de un modelo *bagging*, **establecemos sobre dicho parámetro el número de variables independientes en cada set candidato**.
- *nodesize*: tamaño máximo de nodos finales.
- *sampsizes*: número de observaciones seleccionadas aleatoriamente (con o sin reemplazamiento) para la construcción del árbol.
- *replace*: si el sorteo anterior se realiza con o sin reemplazamiento (por defecto, con reemplazamiento).

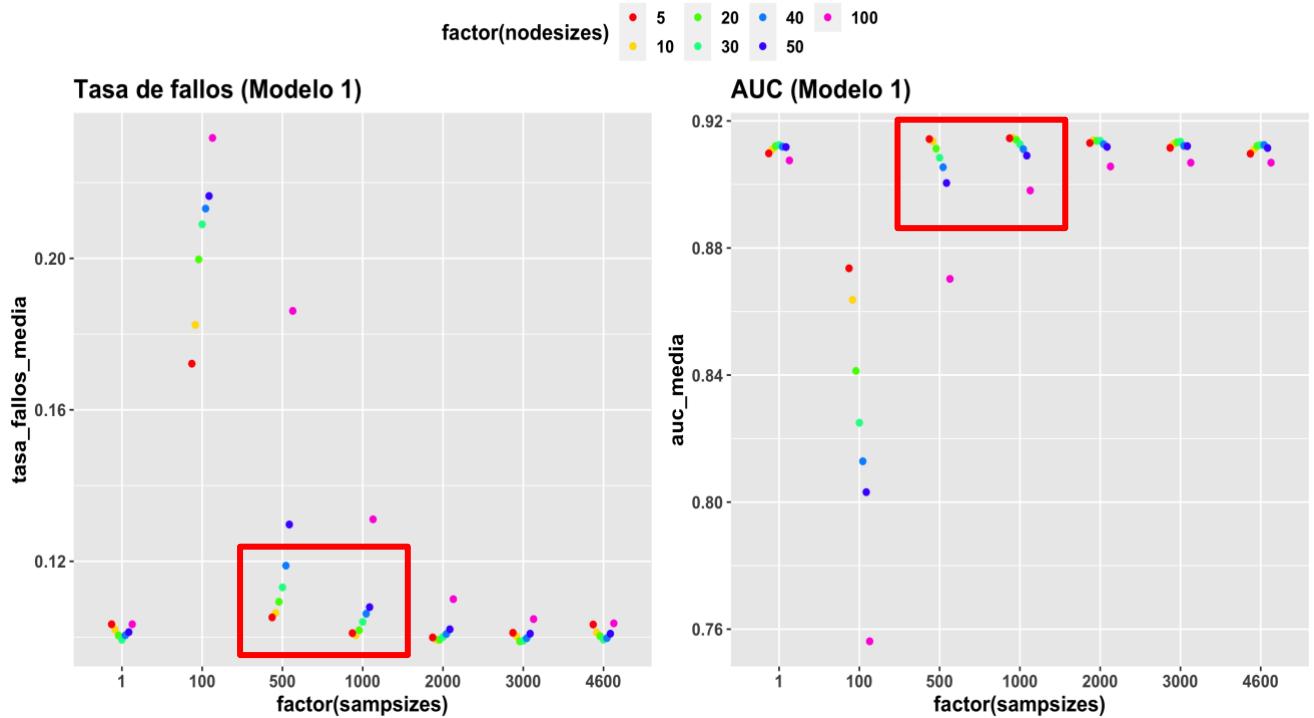
Dado que *mtry* debe corresponder con el número de variables *input* y el número de árboles ya está definido, quedan por tunear tanto *nodesize* como *sampsizes* y *replace*. Por tanto, **comenzamos ajustando tanto el tamaño de la muestra como el tamaño máximo de nodos finales** (en el caso del parámetro *replace*, una vez obtenidos los modelos *bagging* para ambos sets, con el mejor de ambos probamos a seleccionar muestras sin reemplazamiento).

Antes de tunear *sampsizes*, y dado que estamos trabajando con validación cruzada, **dispondremos de menos observaciones para construir el modelo, de forma que debemos establecer un tamaño máximo sobre dicho parámetro**. Concretamente, dado que disponemos de 5854 observaciones y 5 grupos de validación cruzada, cada grupo tendrá disponible  $5854 \times (4/5) \sim 4683$  observaciones, siendo el tamaño máximo de muestra que podemos probar:

```
mtry.1      <- 5
## Redondeamos 4683 a 4600
sampsizes.1 <- list(1, 100, 500, 1000, 2000, 3000, 4600)
nodesizes.1 <- list(5, 10, 20, 30, 40, 50, 100)

bagging_modelo1 <- tuneo_bagging(surgical_dataset, target = target,
                                    lista_continua = var_modelo1,
                                    nodesizes = nodesizes.1,
                                    sampsizes = sampsizes.1, mtry = mtry.1,
                                    ntree = n.trees.1, grupos = 5, repe = 5)
```

A continuación, por cada combinación *nodesize* - *sampsizes* mostramos el promedio tanto de la tasa de fallos como de AUC:



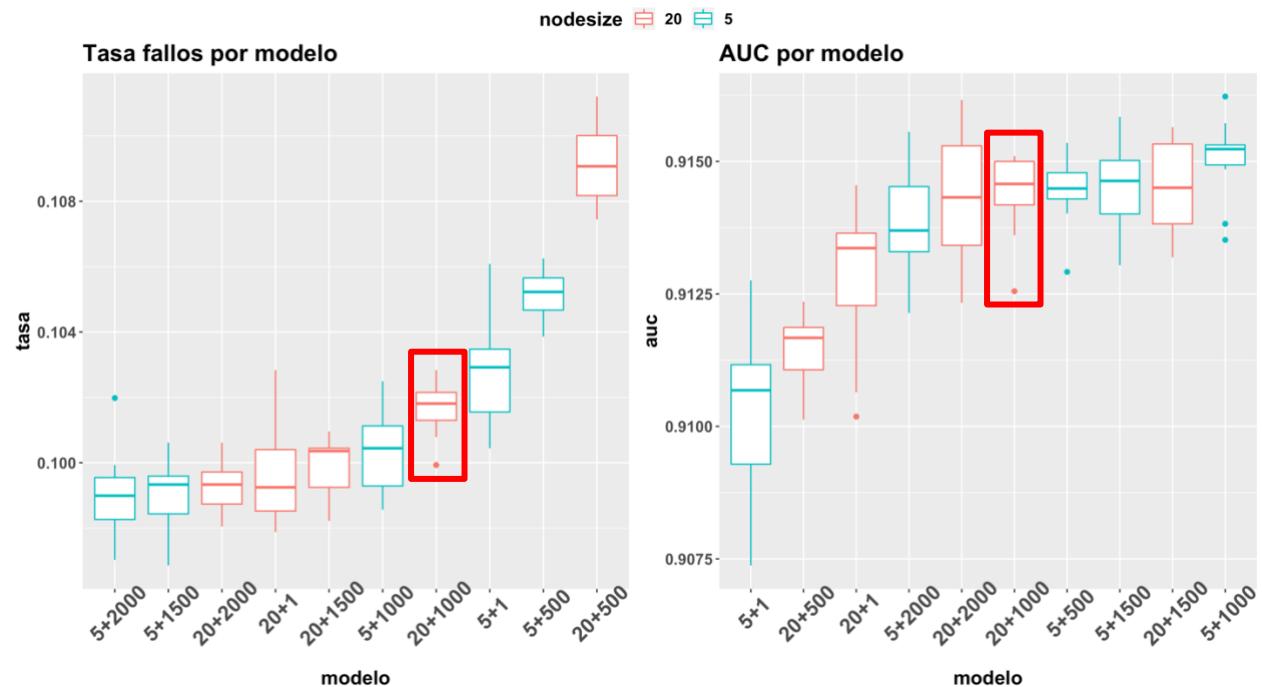
*Figure 26. Distribución tasa de fallos y AUC por sampsizes y nodesize (Modelo 1) (I)*

En un primer análisis, caben destacar fundamentalmente dos aspectos:

1. En relación con *nodesize*, tanto en tasa de fallos como en AUC se obtienen buenos resultados en torno a un valor de 5-30, aproximadamente, pues a partir de 50-100 (árboles menos complejos), la precisión del modelo comienza a disminuir. Sin embargo, ¿Merece la pena aumentar la complejidad del árbol, disminuyendo el parámetro *nodesize*? A simple vista, la diferencia entre un modelo de mayor complejidad (*nodesize* = 5), y un modelo de menor complejidad (*nodesize* = 20, por ejemplo), es muy pequeña (puntos rojo y verde). Por tanto, una posibilidad sería decantarse por un *nodesize* elevado, en torno a 20.
2. Por otro lado, llama la atención el tuneo sobre el parámetro *sampsizes*. Bien es cierto que conforme aumenta el tamaño de la submuestra, mayor es la precisión. No obstante, ¿Qué diferencia existe entre un modelo en el que se utilizan todas las observaciones (*sampsizes* = 1) y un modelo con tan solo 500 o 1000 submuestras? No hay demasiada diferencia, en especial con un *nodesize* en torno a 20, tal y como comentamos en el apartado anterior. De este modo, y gracias además al elevado número de árboles del que disponemos (900), un valor *sampsizes* bajo permite no solo reducir el tiempo de cómputo para entrenar el modelo, sino además una mayor variedad, al sortear menos observaciones por cada árbol.

Es decir, para obtener un buen modelo *bagging* no es necesario emplear todas las observaciones, ni tampoco árboles con demasiada complejidad, sino que con pocas muestras (en torno a 500 o 1000), y un *nodesize* alto (20), se obtienen muy buenos resultados.

Por tanto, una vez realizado el primer análisis, probamos a variar (con 10 repeticiones), tanto un modelo de *bagging* de mayor complejidad (*nodelsize* = 5) y de menor complejidad (20), así como el tamaño de la muestra, probando a sortear hasta 2000 observaciones:

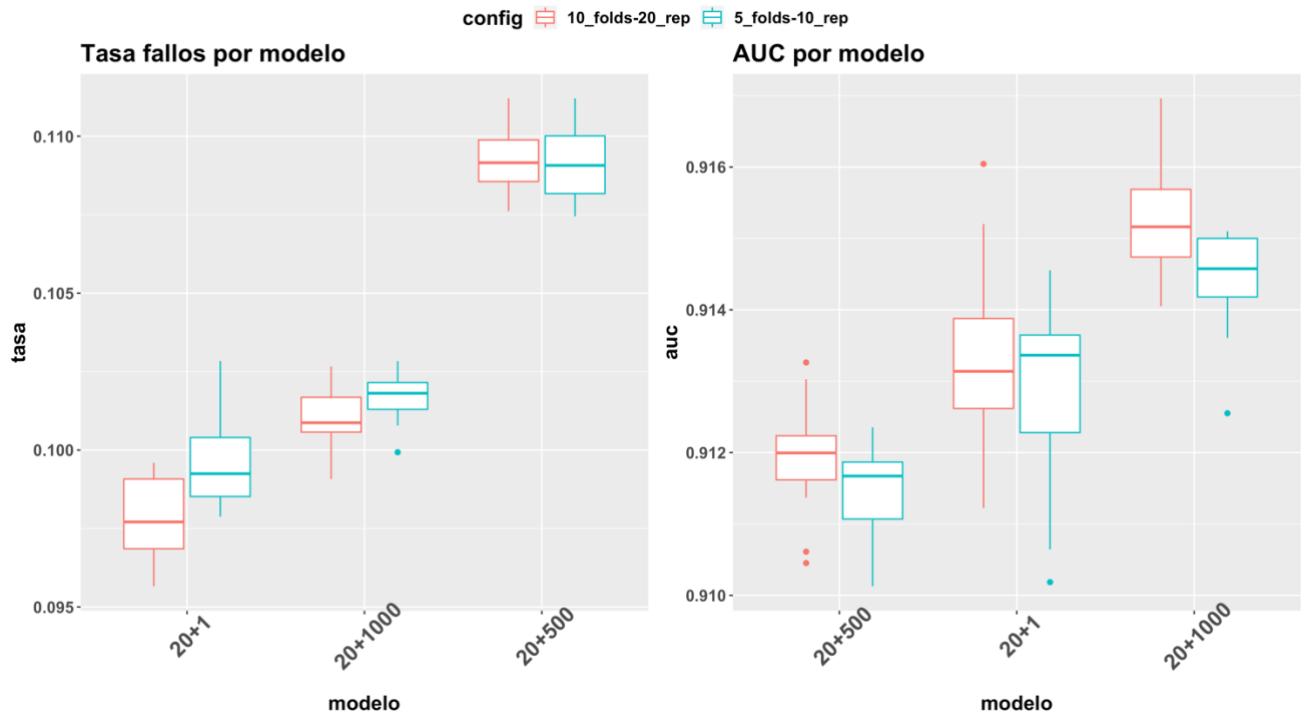


**Figure 27. Distribución tasa de fallos y AUC por nodesize y sampsize (Modelo 1) (II)**

Nuevamente, sin fijarnos en la escala del eje, una posible opción por la que decantarse sería la de un modelo con *nodesize* 5 y *sampsize* 1000, pues presenta el mayor valor AUC y una tasa de fallos baja. No obstante, los valores del eje pueden llevarnos nuevamente a engaño. Por ejemplo, **la diferencia entre este modelo y uno mucho más sencillo como es el caso de *nodesize* 20 y *sampsize* en torno a 500-1000 (20 + 500 o 20 + 1000) es muy pequeña** (hablamos de un sesgo en torno a 0.10 y un AUC de 0.91, por lo que la diferencia es del orden de milésimas).

Incluso con un menor número de muestras, la varianza de los modelos se ve reducida en comparación con todas las observaciones, aunque la diferencia (por las escalas de los ejes) sea menor.

Por tanto, dada la poca ganancia que presentan los modelos complejos, **nos decantamos por un modelo *bagging* con *nodesize* moderado/alto (20) y un tamaño de muestra en torno a 500 y 1000**. Sobre ambos candidatos, para controlar mejor el tamaño óptimo de la muestra, probamos con una validación cruzada de 10 grupos y 20 repeticiones:



*Figure 28. Distribución tasa de fallos y AUC (Modelo 1), aumentando grupos y repeticiones*

Por lo general, incluso aumentando el número de grupos y repeticiones, el orden de los modelos se mantiene idéntico, tanto en tasa de fallos como en AUC. Por tanto, de los dos posibles modelos candidatos (con *sampsize* 500 o 1000), aunque la diferencia entre ambos, así como el sesgo y varianza no sean muy significativas (dada la escala de los ejes), **nos decantamos por un modelo con un tamaño moderado, de 1000 submuestras**, es decir, con tan solo el 1000 / 5854 ~ 17 % de las observaciones, reduciendo la tasa de fallos por debajo del 10 %.

### 7.3 Modelo 2

Del mismo modo, realizamos los mismos pasos con el segundo *set* de cuatro variables, comenzando con el tuneo tanto de *nodesize* como de *sampsize*, obteniendo el promedio en tasa de fallos y AUC con 5 repeticiones:

```
mtry.2      <- 4
sampsizes.2 <- list(1, 100, 500, 1000, 2000, 3000, 4600)
nodesizes.2 <- list(5, 10, 20, 30, 40, 50, 100)

bagging_modelo1 <- tuneo_bagging(surgical_dataset, target = target,
                                    lista_continua = var_modelo2,
                                    nodesizes = nodesizes.2,
                                    sampsizes = sampsizes.2, mtry = mtry.2,
                                    ntree = n.trees.2, grupos = 5, repe = 5)
```

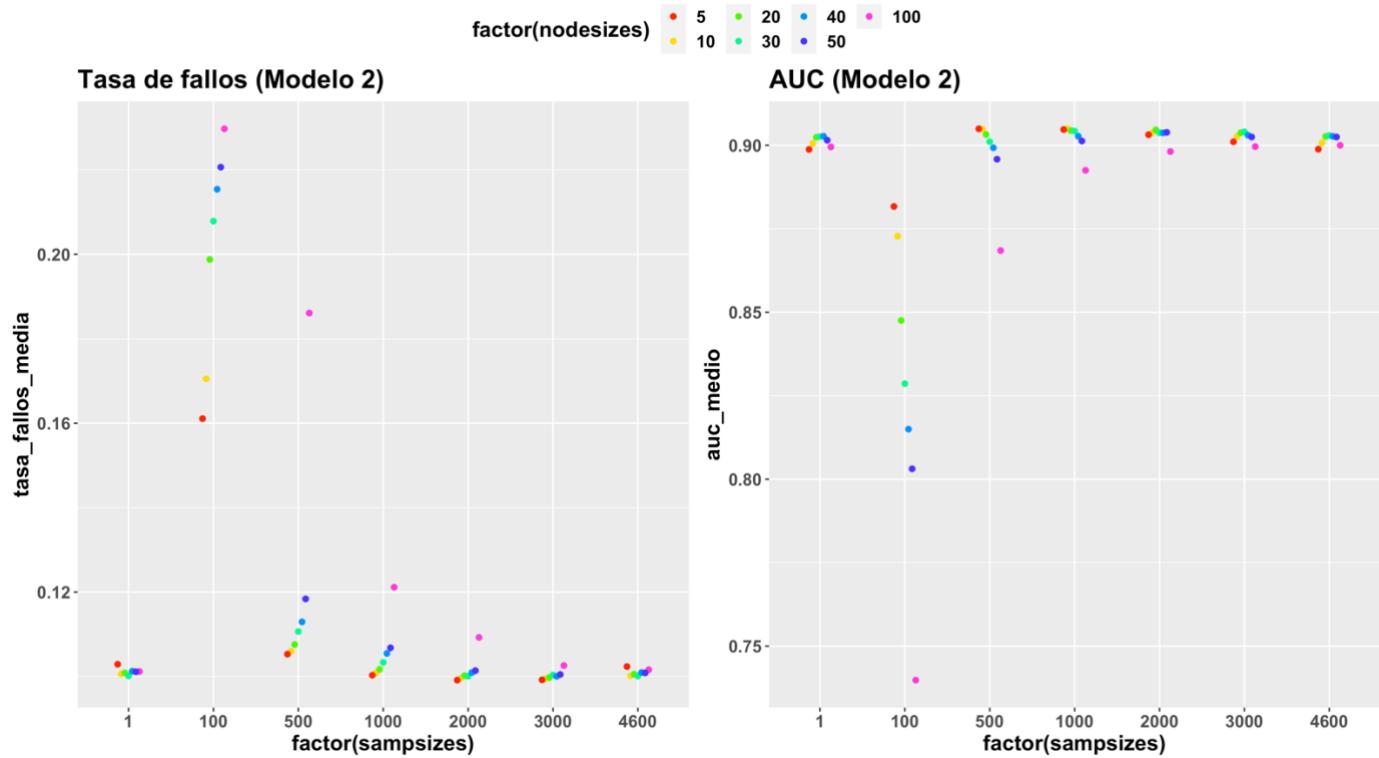
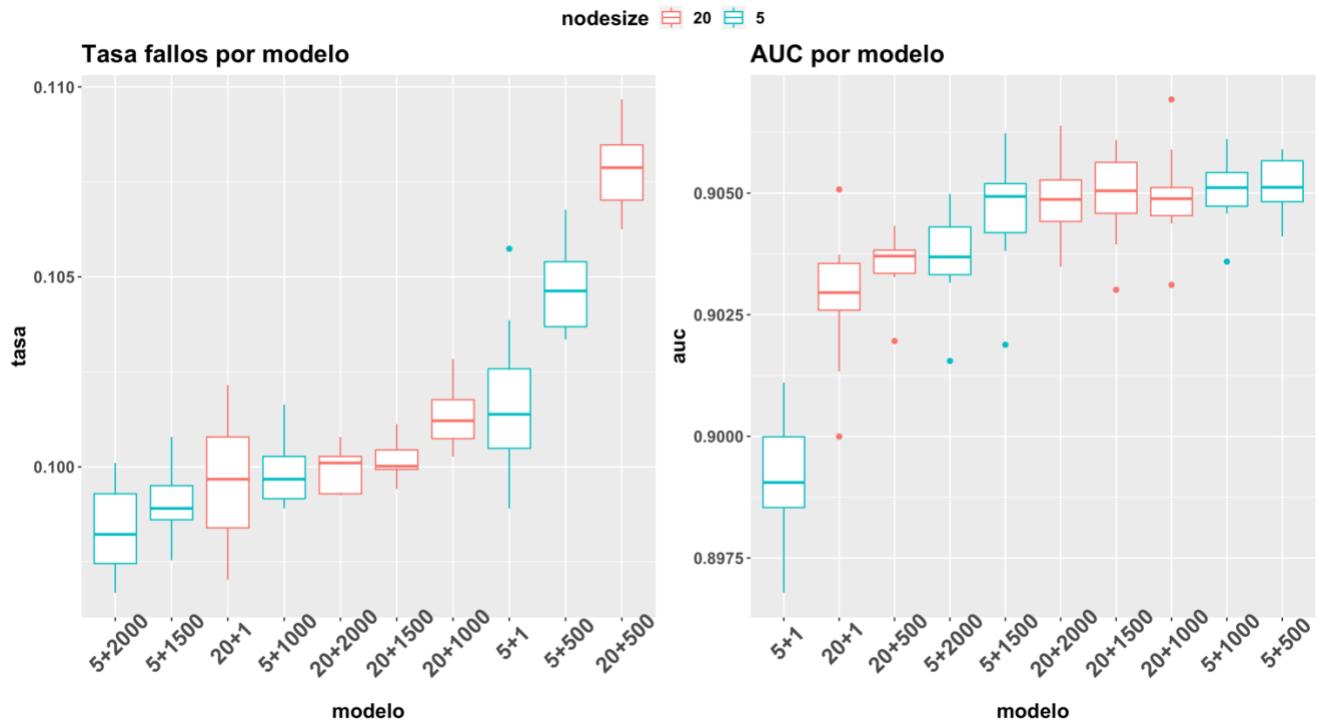


Figure 29. Distribución tasa de fallos y AUC por sampsizes y nodesize (Modelo 2) (I)

Nuevamente, nos encontramos con un comportamiento similar al obtenido con el primer modelo: por un lado, con un valor *nodesize* moderado/alto (en torno a 20) se obtienen buenos resultados, por lo que no es necesario utilizar árboles demasiado complejos, pues la ganancia tanto en tasa de fallos como en AUC es ínfima. Por otro lado, y en relación con *sampsizes*, **no es necesario emplear todas las observaciones**, sino que con tan solo 500 o 1000 muestras, el modelo obtiene resultados similares a utilizar todo el *dataset*.

Por tanto, y del mismo modo que en el primer *set* de variables, **comparamos la diferencia entre un modelo de mayor complejidad (*nodesize* = 5) y de menor complejidad (*nodesize* = 20), utilizando diferentes tamaños de *sampsizes***:

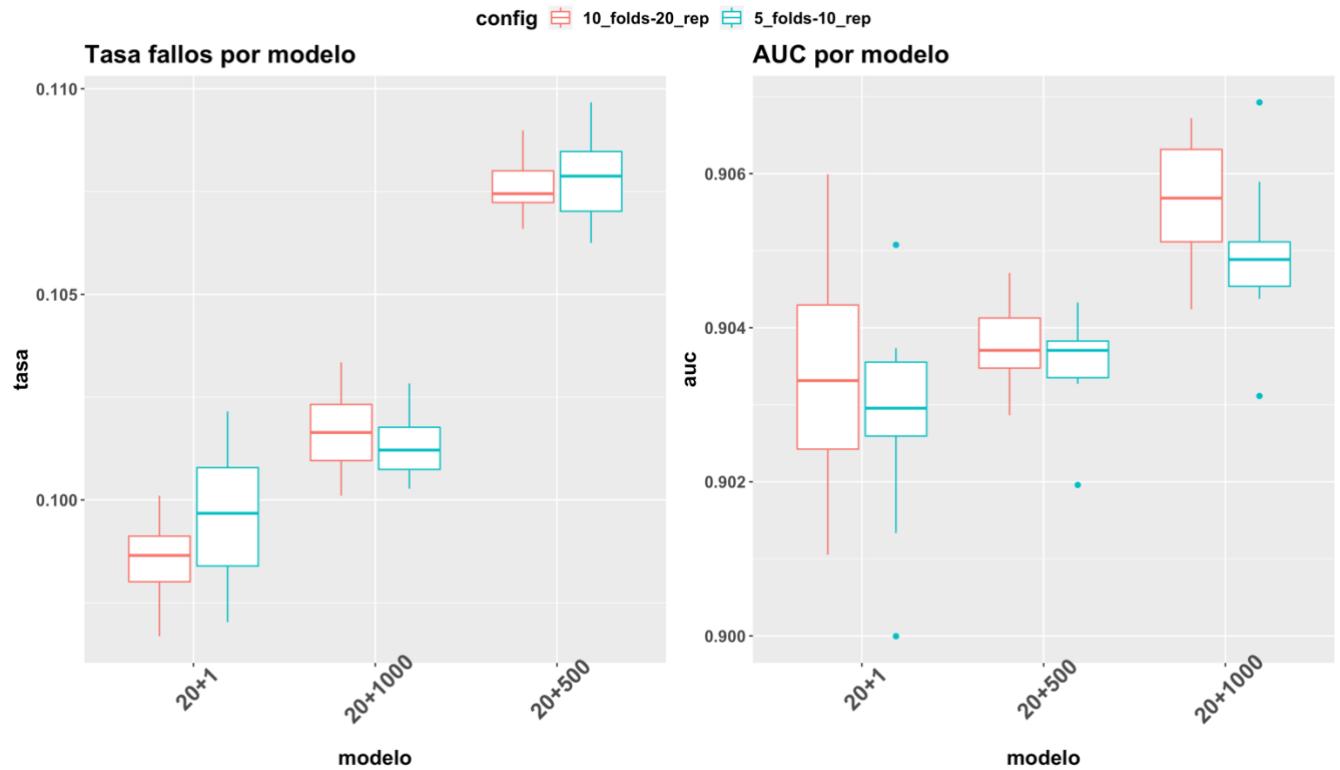
```
nodesizes.2 <- list(5, 20)
## Probamos con un sampsizes entre 500 y 2000 observaciones (1 = todas las observaciones)
sampsizes.2 <- list(1, 500, 1000, 1500, 2000)
bagging_modelo1_v2 <- tuneo_bagging(surgical_dataset, target = target,
lista_continua = var_modelo2,nodesizes = nodesizes.2,
sampsizes = sampsizes.2, mtry = mtry.2,ntree = n.trees.2, grupos = 5, repe = 10)
```



*Figure 30. Distribución tasa de fallos y AUC por sampsizes y nodesize (Modelo 2) (II)*

De nuevo, el hecho de aumentar la complejidad de los árboles (reduciendo el valor de *nodesize*), no supone una mejoría relevante al modelo. A modo de ejemplo, la diferencia en la tasa de fallos entre un modelo con *nodesize* 5 (5 + 2000) y un modelo con *nodesize* 20 (20 + 2500), es de apenas unas milésimas de diferencia (del mismo modo sucede con AUC). Es decir, la diferencia entre árboles de menor y mayor complejidad es prácticamente nula, pues la diferencia entre ambos colores de cajas oscila apenas en torno a milésimas.

Por tanto, y del mismo modo que en el primer *set* de variables, nos decantamos por un *nodesize* de 20 y un tamaño de muestra pequeño/moderado, en torno a 500-1000 observaciones. Sobre ambos candidatos, probamos de nuevo con una validación cruzada de 10 grupos y 20 repeticiones, comprobando de este modo la estabilidad de la varianza:



*Figure 31. Distribución tasa de fallos y AUC (Modelo 2), aumentando grupos y repeticiones*

Incluso aumentando el número de grupos, el orden y varianza de los modelos se mantiene prácticamente igual, aunque aumenta ligeramente al utilizar el *dataset* completo. En conclusión, nos decantamos por la misma configuración que con el primer *set de variables*: *nodesize* 20 y *sampszie* 1000.

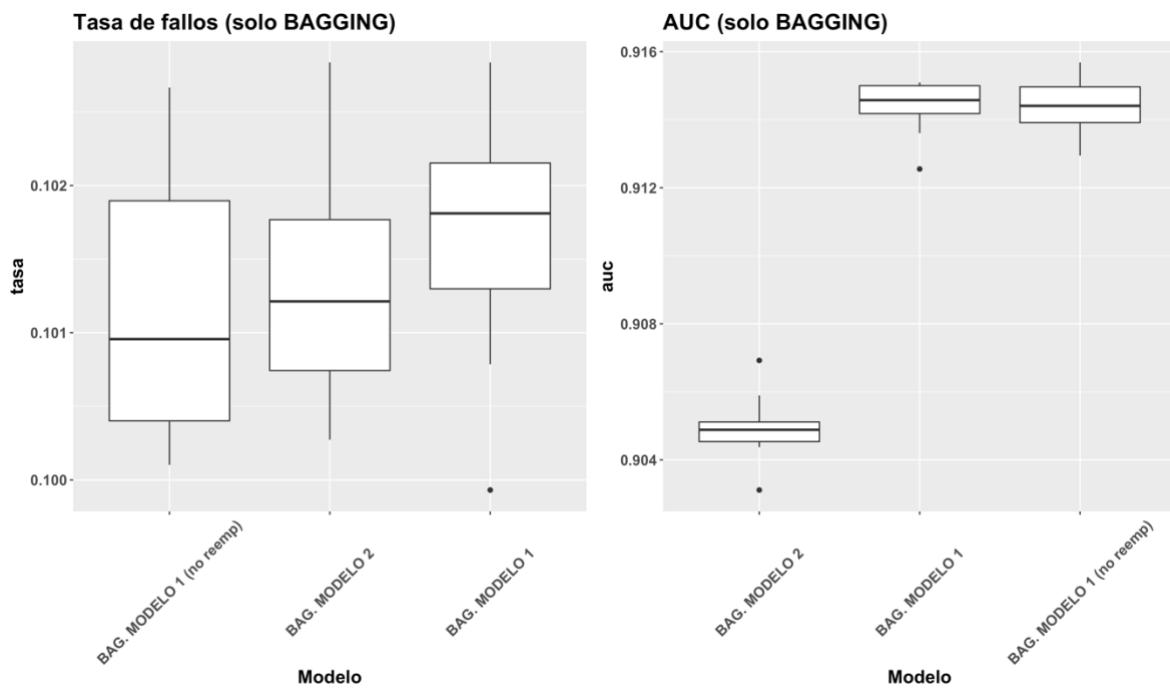
#### **RESUMEN *bagging*:**

1. Modelo 1: *nodesize* = 20 y *sampszie* = 1000.
2. Modelo 2: *nodesize* = 20 y *sampszie* = 1000.

#### **7.4 Modelo sin reemplazamiento**

Una vez tuneados ambos *sets* de variables, veamos la diferencia entre un modelo con *sampszie* con reemplazamiento y sin reemplazamiento, por mera curiosidad:

```
#-- ¿Y si Lo probamos sin reemplazamiento? Probamos con el mejor modelo en terminos de AU C (modelo 1)
bagging_modelo_sin_reemp <- tuneo_bagging(surgical_dataset, target = target,
                                             lista_continua = var_modelo1, nodesizes = 20,
                                             sampsizes = 1000, mtry = mtry.1, ntree = n.trees.1,
                                             grupos = 5, repe = 10, replace = FALSE)
```

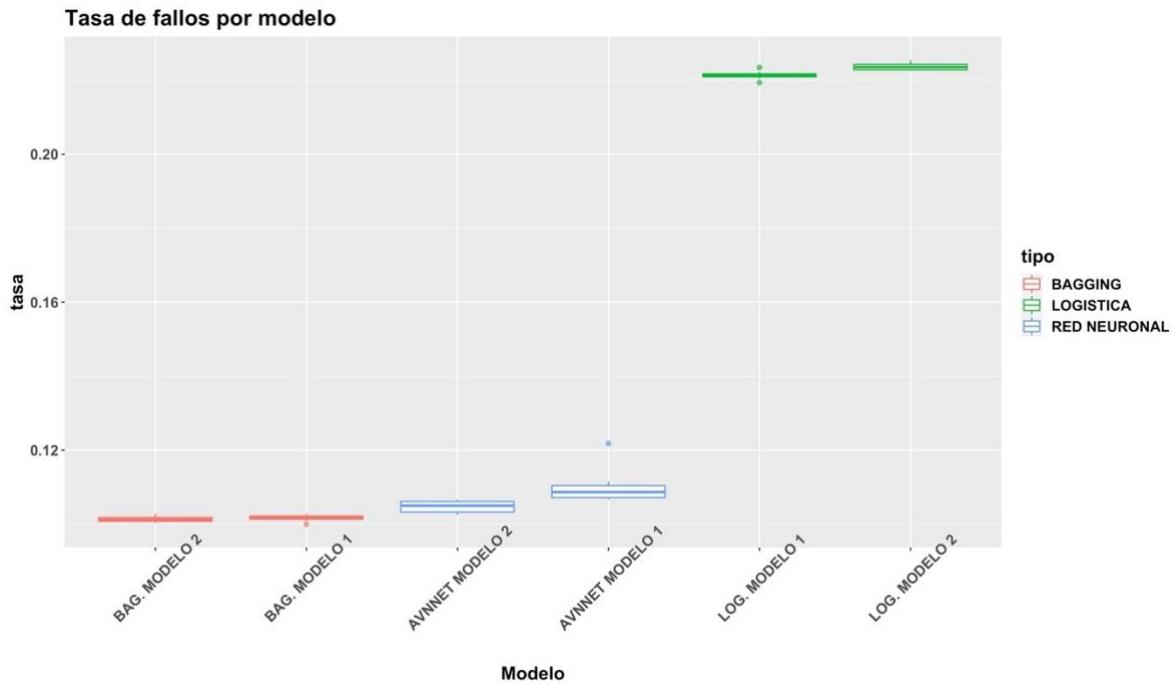


**Figure 32. Distribución tasa de fallos y AUC con y sin reemplazamiento**

A simple vista, no existe diferencia entre modelos con o sin reemplazamiento en las observaciones.

## 7.5 Comparación final

Finalmente, realizamos la comparación final de ambos modelos *bagging*:



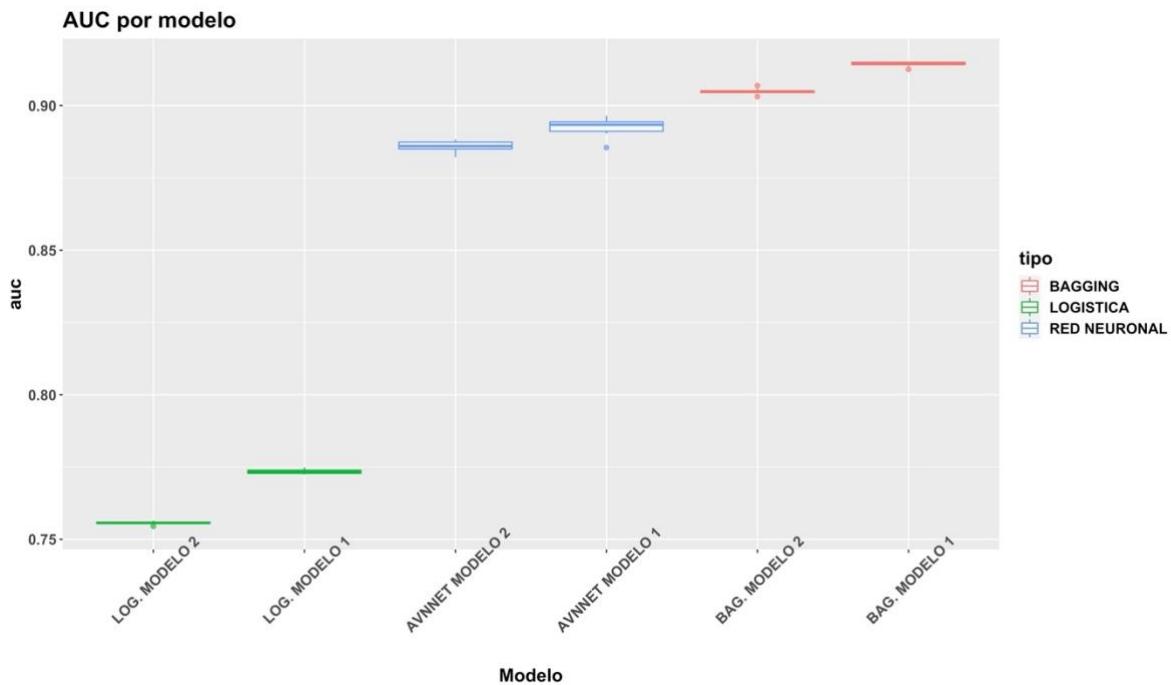


Figure 33. Comparación logística-avnnet-bagging

De nuevo, la relación no lineal entre las variables *input* se ve reflejado en los resultados obtenidos tanto en modelos de red como *bagging*, siendo este último caso el que obtiene mejores resultados tanto en tasa de fallos como en AUC (llegando hasta 0.9), aunque la diferencia, dada la escala de los ejes, no sea muy relevante.

Además, cabe recalcar la **poca o escasa diferencia** entre ambos *sets* de variables candidatos. De hecho, incluyendo la variable adicional *ccsMort30Rate* (modelo 1), la tasa de fallos en cualquier modelo continúa siendo prácticamente idéntica al del modelo 2 con solo 4 variables *input*; mientras que el AUC apenas se ve mejorado en tan solo 0.01.

## 8. Random Forest

### 8.1 Selección del número de árboles y *mtry*

Una vez elaborados los modelos *bagging*, y dado que *Random Forest* si realiza el sorteo de variables por cada división del árbol **¿Con 900 árboles es suficiente?** Para comprobarlo, analicemos en ambos *sets* la evolución del *ratio* de error al variar el parámetro *mtry*:

### 8.1.1 Modelo 1

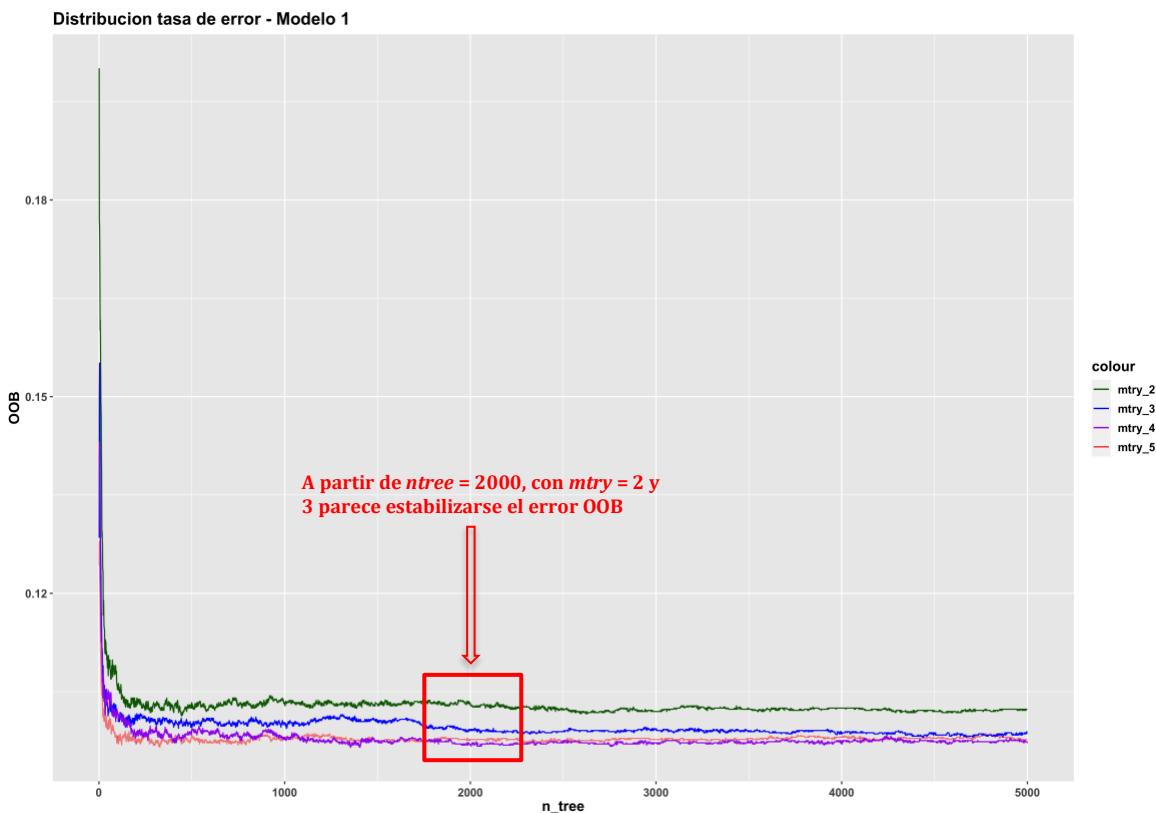


Figure 34. Error rate (Modelo 1) en función de mtry

En primera instancia, observamos que el error (para cualquier  $mtry$ ) se estabiliza prácticamente a partir de 2000 árboles, momento en el que el error *Out of bag* es estable con  $mtry = 2$  y  $3$  (especialmente en este último caso). Sin embargo, con otros valores  $mtry$ , es decir, **sorteando un mayor número de variables**, el error se estabiliza con antelación:

- $mtry = 4$ : a partir de 1500 árboles, aproximadamente.
- $mtry = 5$ : a partir de 900 árboles (caso del modelo *bagging*).

Tras una primera impresión, y utilizando 2000 árboles, **lanzamos los primeros modelos random forest**, utilizando los diferentes valores  $mtry$ . En relación con el resto de los parámetros como *nodesize* o *sampsizes*, **utilizamos los empleados en el modelo bagging**: 20 + 1000.

```
#-- 2000: numero de arboles a partir del cual se estabiliza con mtry = 2 y 3
# Inicialmente, probamos con 5 repeticiones
```

```
mtry.1 <- c(2,3,4,5) # mtry = 5 -> modelo bagging
primera.imp <- tuneo_rf(surgical_dataset, target=target, lista.continua=var_mode
lo1, grupos=5, repe=5, nodesizes=20, mtry=mtry.1, ntree=2000, replace=TRUE,
sampsizes=1000)
```

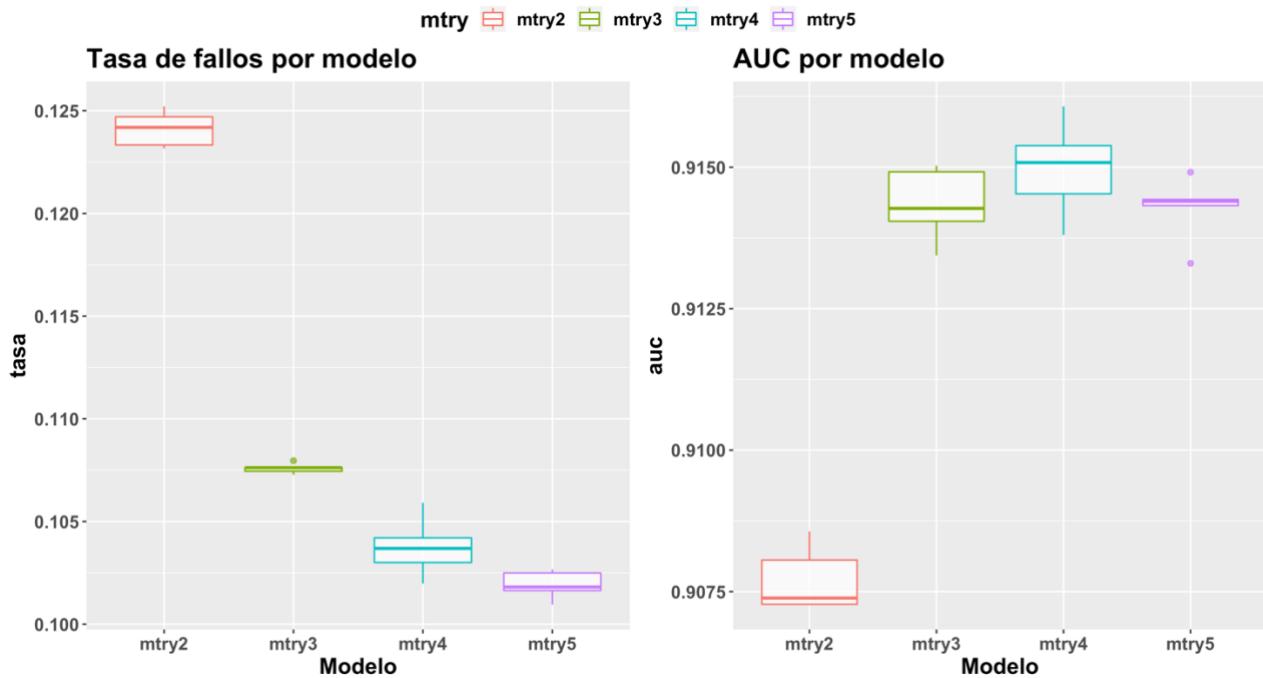


Figure 35. Tasa de fallos y AUC en función de mtry (Modelo 1)

A simple vista, podemos comprobar que no es realmente necesario sortear todas las variables en cada nivel del árbol, sino que con un tamaño menor, moderado, en torno a 3, se obtienen resultados muy similares, aunque con una ganancia en la tasa de fallos muy pequeña (0.107-0.108 frente a 0.102 del modelo *bagging*). Por otro lado, aunque teóricamente sortear menos variables permite controlar mucho mejor la varianza del modelo, en este caso (y dada la escala de ambos ejes), no es el principal inconveniente en este conjunto de datos, pues sorteando 3, 4 o 5 variables la varianza se mantiene prácticamente igual. Por el contrario, sorteando tan solo 2 variables, aunque el AUC no disminuye demasiado (de 0.915 a 0.907), la tasa de fallos si se ve afectada en mayor medida, aumentando del 10 al 12.5 %.

Por tanto, en vista a los resultados obtenidos, nos decantamos por un valor mtry moderado, en torno a 3, dado que sorteando dos variables la tasa de fallos comienza a aumentar hasta 0.12, aunque la ganancia de error no sea demasiado alta.

### 8.1.2 Modelo 2

Del mismo modo, realizamos los mismos pasos para el segundo *set* de variables candidato, comenzando con el número de árboles:

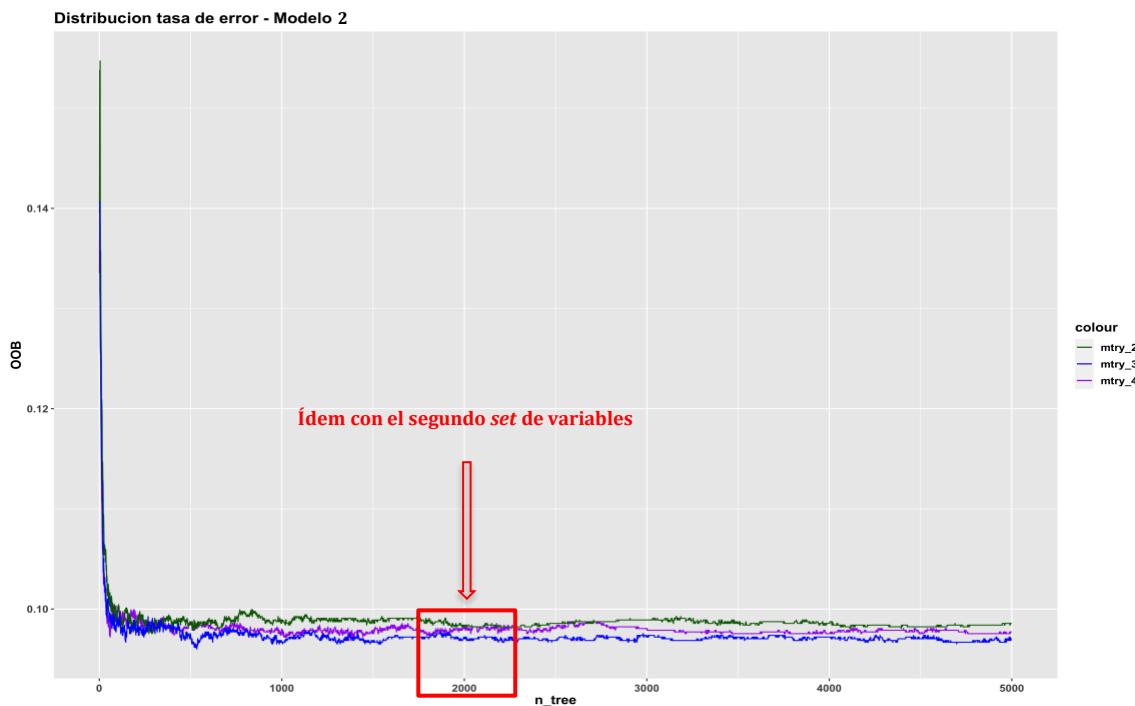


Figure 36. Error rate (Modelo 2) en función de mtry

Al igual que en el primer modelo, a excepción de  $mtry = 4$  (correspondiente con el modelo *bagging*), para estabilizar el error **se requieren un mayor número de árboles**, en torno a 2000 aproximadamente tanto para  $mtry = 2$  como  $mtry = 3$ . A continuación, echemos un primer vistazo al comportamiento de los modelos, tanto en sesgo como en varianza, utilizando los diferentes valores  $mtry$ , así como 2000 árboles y el mejor valor  $mtry$  y *nodesize* obtenidos en *bagging* para el segundo set de variables (20 + 1000):

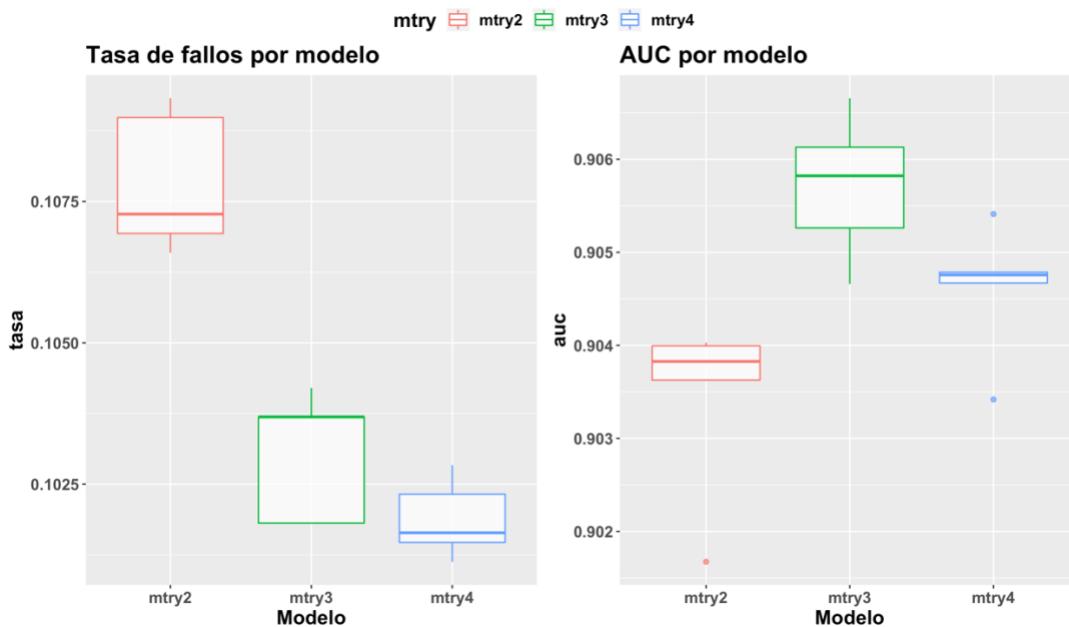


Figure 37. Tasa de fallos y AUC en función de mtry (Modelo 2)

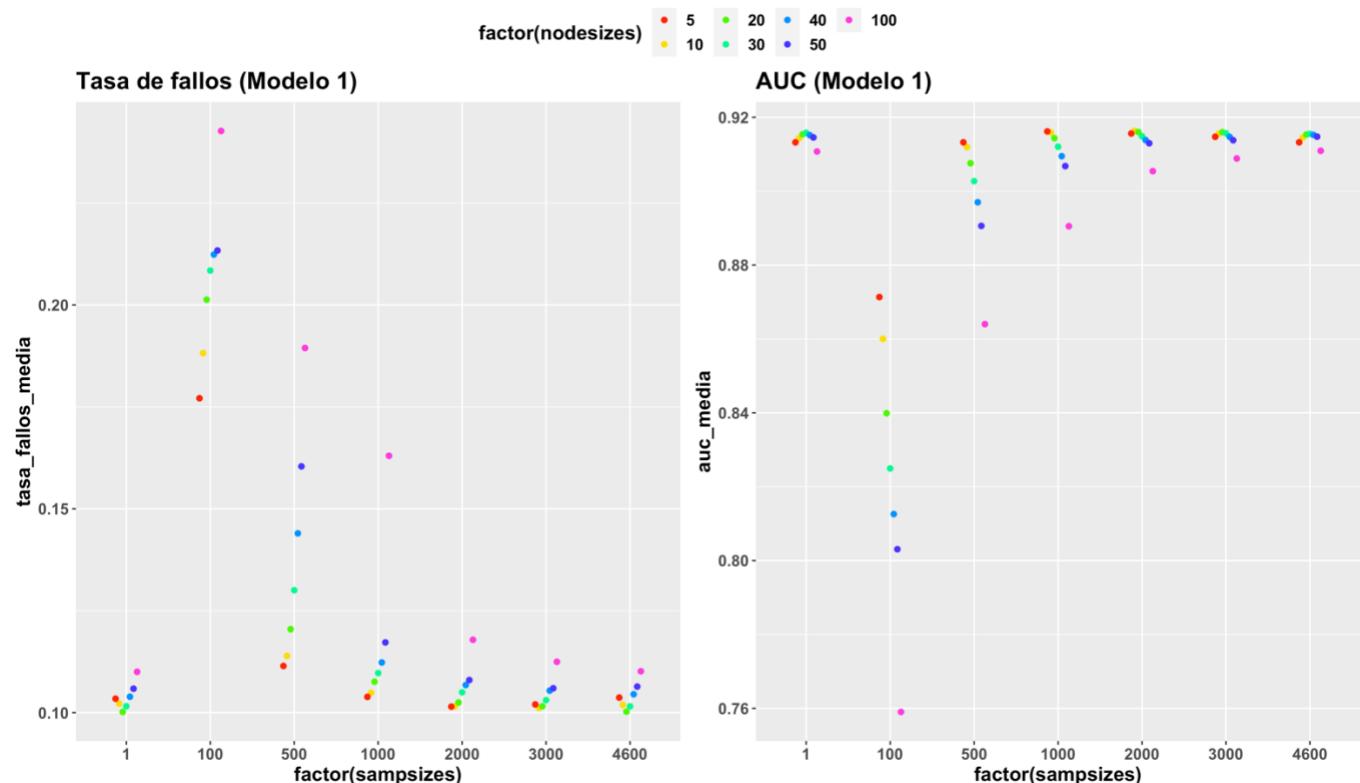
En primera instancia, una posibilidad sería decantarse por un modelo con  $mtry = 3$ , sin llegar a utilizar todas las variables, dado el mayor AUC. Sin embargo, y en base a la escala del eje Y, **el error y varianza que presenta el modelo con  $mtry = 2$  no es tan pronunciado**:

1. En el caso de la tasa de fallos, la diferencia es de 0.005 (de 0.1025 en el caso de  $mtry = 4$  a 0.1075 en el caso de  $mtry = 2$ ). Además, la varianza que presenta en torno al modelo, teniendo en cuenta la escala del eje, es del orden de milésimas, pese a que el diagrama de cajas "dá la sensación" de una alta variabilidad en los resultados.
2. En el caso del área bajo la curva ROC, la diferencia también se sitúa en torno a las milésimas (de 0.906 en el caso de  $mtry = 4$  a 0.904 en el caso de  $mtry = 2$ ).

Dicha diferencia puede ser debida a numerosos factores: estructura de remuestreo, selección de observaciones con *sampsizes*, valor de la semilla etc. Por tanto, dado que la diferencia no es apenas relevante, con el segundo set de variables **nos decantamos por un valor  $mtry = 2$** .

## 8.2 Modelo 1

En un primer comienzo, con un valor  $mtry = 3$  y el mismo *nodelsize* y *sampsizes* que en el modelo *bagging*, obtenemos muy buenos resultados. No obstante, hemos considerado un valor de *nodelsize* y *sampsizes* por defecto, esto es, los mejores parámetros obtenidos en *bagging*. Por tanto, **¿Puede llegar a ser influyente el hecho de aumentar o reducir el valor *nodelsize*, o más importante aún, el tamaño de la submuestra en *sampsizes*?** Del mismo modo que en *bagging*, analicemos la importancia de ambos parámetros, tuneando el modelo con diferentes valores (comenzando con tan solo 5 repeticiones):



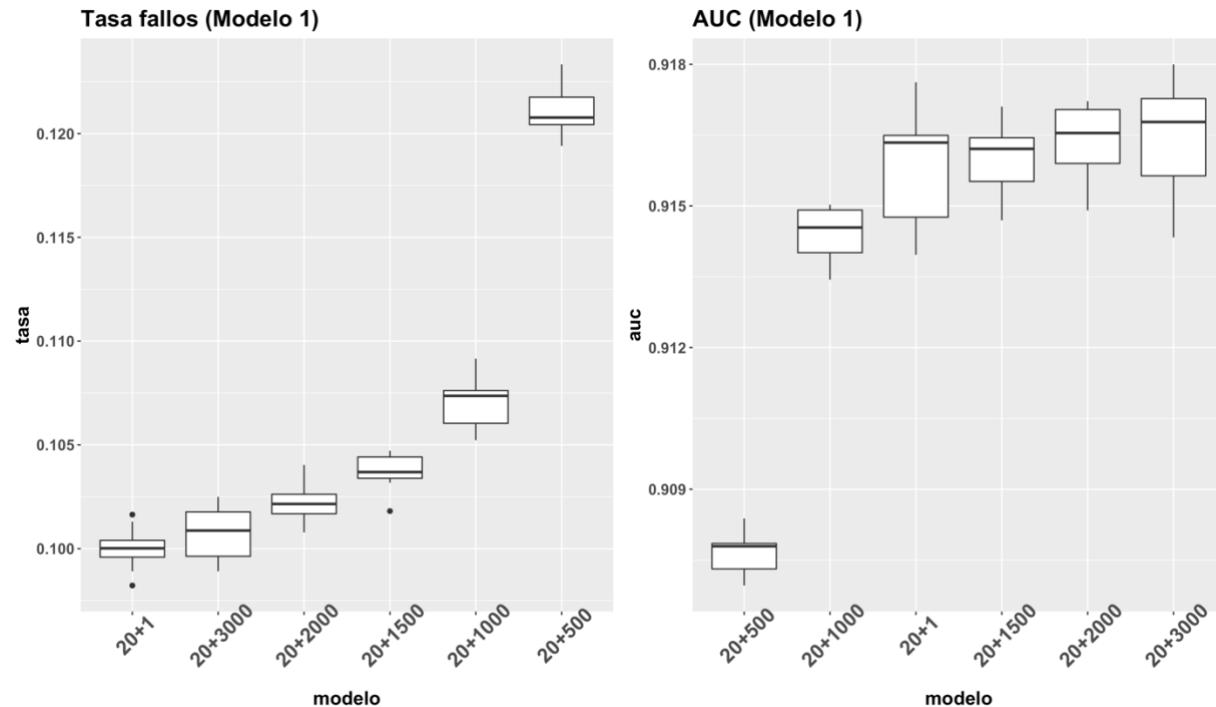
*Figure 38. Tasa de fallos y AUC en función de nodelsize y sampsizes (Modelo 1)*

Analizando el promedio de cada modelo, detectamos prácticamente las mismas características que en el modelo *bagging*:

1. **Aumentando la complejidad del árbol, con un *nodesize* menor, la ganancia que supone tanto en tasa de fallos como en AUC no es muy relevante**, por lo que podemos emplear un valor de 20 como tamaño mínimo de cada nodo.
2. Además, como primera impresión **un valor *sampsizes* en torno a 500-1000 vuelve a ser una buena opción por la que decantarse, obteniendo los mismos resultados que con el conjunto total de las observaciones**, lo que permite la construcción de árboles mucho más diferentes entre si gracias al sorteo de menos muestras. No obstante, analicemos más en detalle tanto el sesgo como la varianza al variar el tamaño de la muestra (manteniendo el valor *nodesize* a 20 y aumentando el número de repeticiones a 10):

```
nodesizes.1 <- list(20)
# Probamos con un tamaño sampsizes entre 500 y 3000 (1 = todas Las observaciones)
sampsizes.1 <- list(1, 500, 1000, 2000, 3000)

bagging_modelo1_mtry3 <- tuneo_rf(surgical_dataset, target = target,
                                      lista_continua = var_modelo1,
                                      nodesizes = nodesizes.1,
                                      sampsizes = sampsizes.1, mtry = 3,
                                      ntree = 2000, grupos = 5, repe = 10)
```



*Figure 39. Tasa de fallos y AUC en función de sampsizes (Modelo 1)*

Incluso sorteando variables, los resultados obtenidos no difieren demasiado de un modelo **bagging tradicional**: mientras que con 500 observaciones la tasa de fallos aumenta hasta 0.12 y el AUC se reduce a 0.90, a partir de un tamaño de 1000 muestras (lo que supone alrededor del 17 % del total de las observaciones), **no solo iguala en AUC al resto de modelos con**

*sampsizes* mayores (en torno a 0.91), sino que además la ganancia de error es de tan solo unas milésimas: entre un *sampsize* = 1000 y 2000-3000 observaciones, la diferencia es de  $0.107 - 0.102 = 0.005$ .

Es decir, **aumentar demasiado el tamaño de las submuestras no aporta una mejoría significativa al modelo**, por lo que nos decantamos nuevamente por un tamaño de 1000.

Por último, para observar en mejor medida el efecto del parámetro *sampsize*, aumentamos a 10 grupos y 20 repeticiones, tanto con 1000 muestras como empleando todas las observaciones:

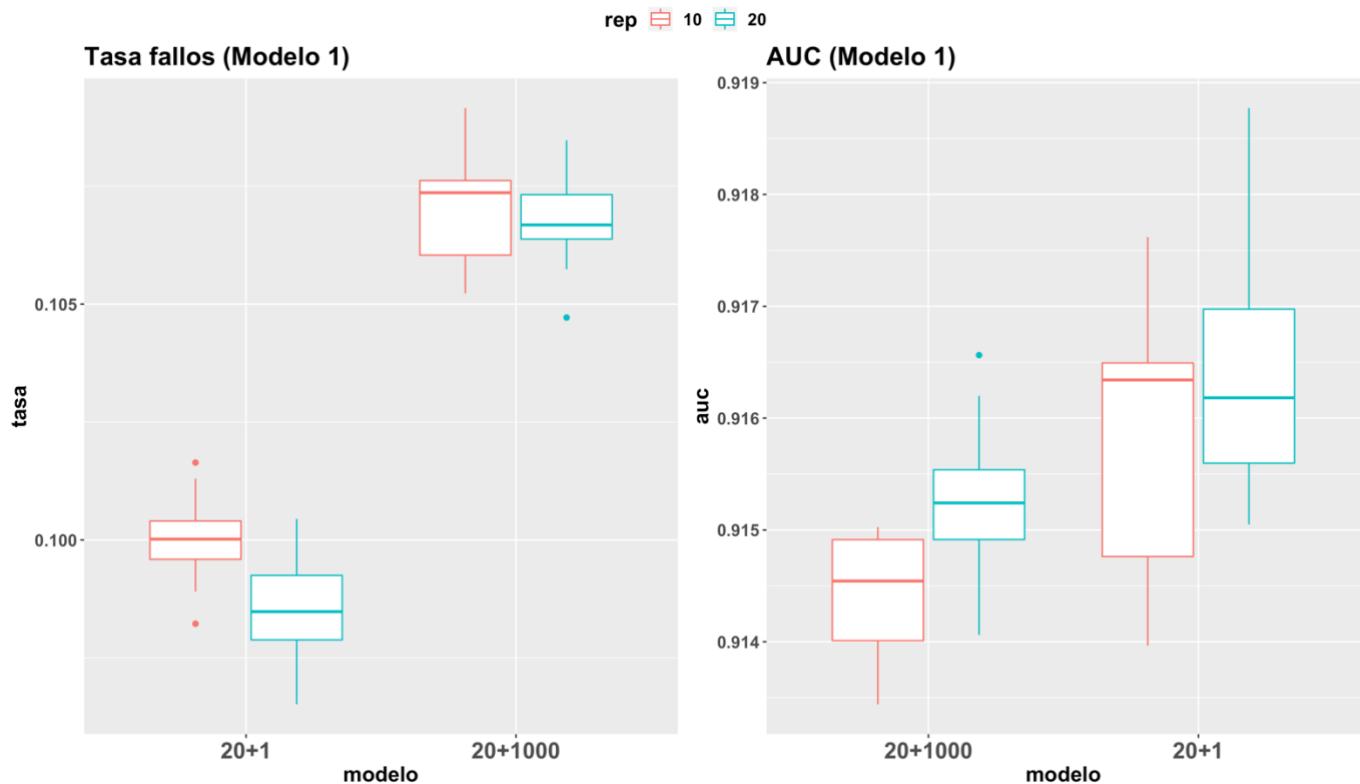


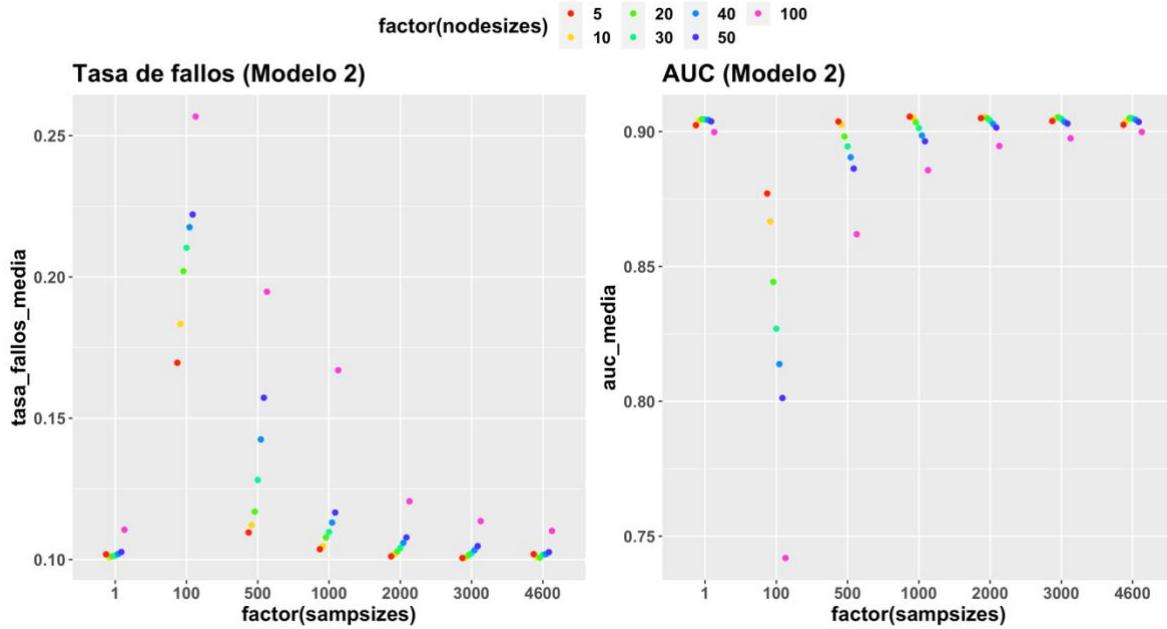
Figure 40. Tasa de fallos y AUC, aumentando a 10 grupos y 20 repeticiones (Modelo 1)

Incluso aumentando a 10 grupos y 20 repeticiones, tanto el sesgo como la varianza se mantienen prácticamente idénticos, salvo pequeñas variaciones del orden de apenas unas milésimas. Por tanto, *nodelsize* 20 + *sampsize* 1000 continúa siendo una buena alternativa, frente a utilizar todas las observaciones.

## 8.2 Modelo 2

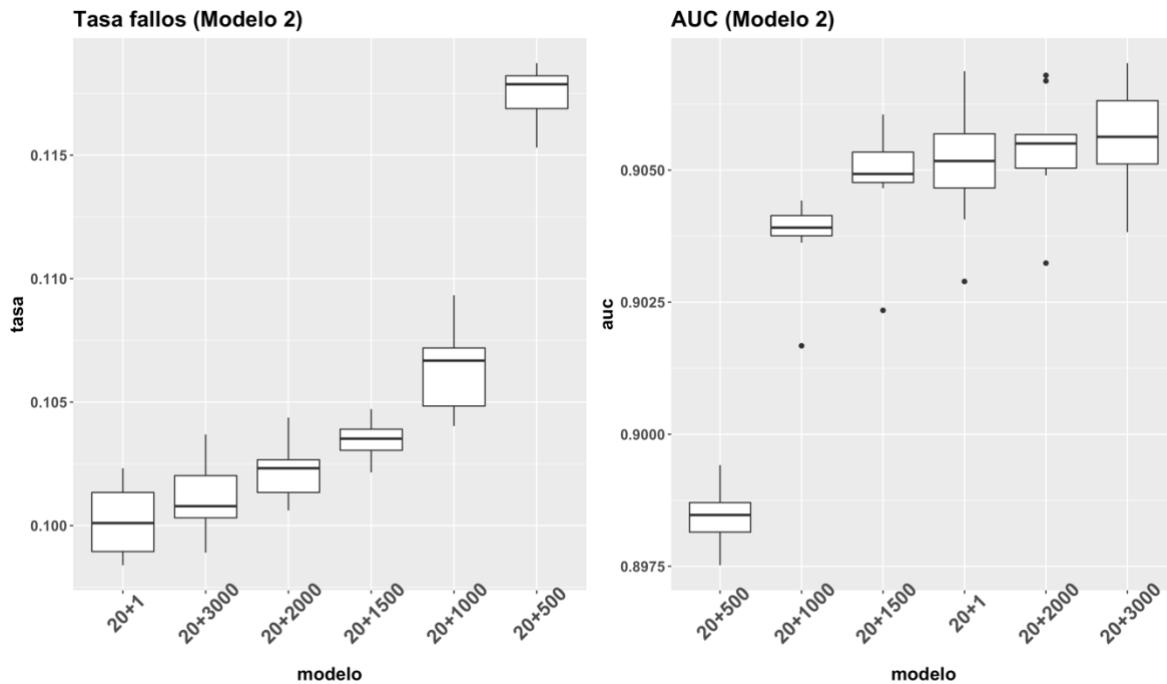
A continuación, realizamos los mismos pasos con el segundo *set* de variables, parametrizando en primer lugar tanto *nodelsize* como *sampsize*:

```
sampsizes.2 <- list(1, 100, 500, 1000, 2000, 3000, 4600)
nodelsizes.2 <- list(5, 10, 20, 30, 40, 50, 100)
```



*Figure 41. Tasa de fallos y AUC en función de nodesize y sampsize (Modelo 2)*

Del mismo modo que sucede con el primer *set* de variables, tanto un **nodesize** en torno a 20 como un valor **sampsize** entre 500 y 1000, parece una buena opción. Además, si analizamos tanto el sesgo como la varianza al variar este último parámetro, aumentando a 10 el número de repeticiones:

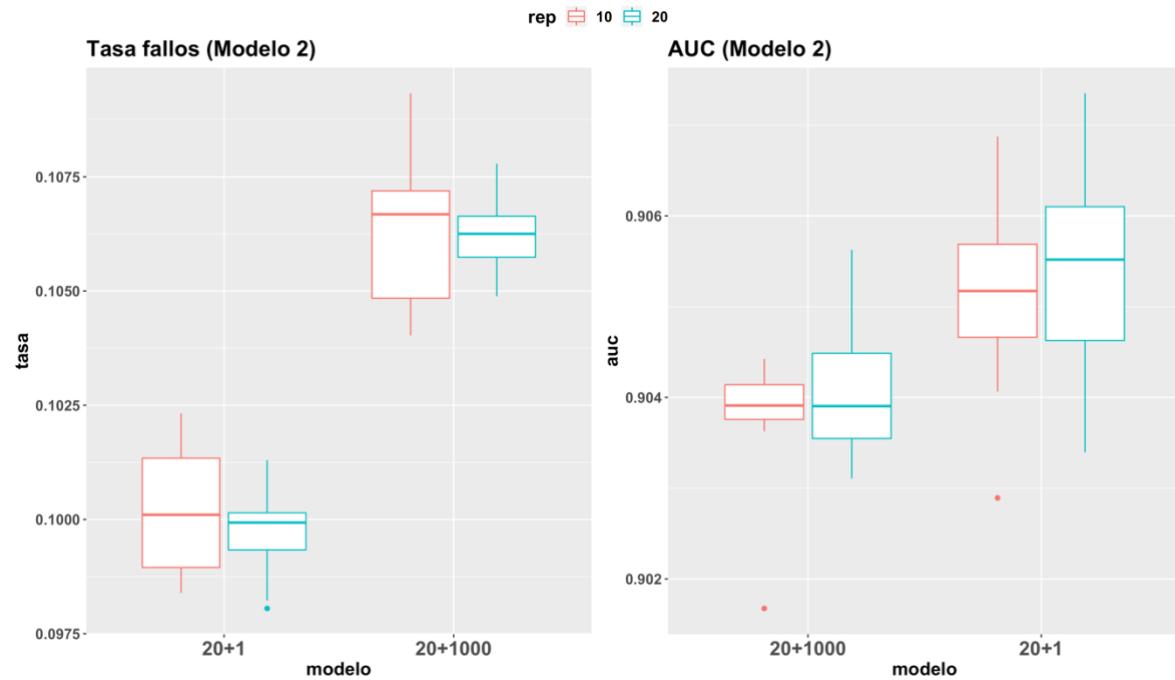


*Figure 42. Tasa de fallos y AUC en función de sampsize (Modelo 2)*

Tenemos el mismo comportamiento que el primer *set* de variables: a partir de un tamaño en torno a 1000 observaciones, la diferencia tanto en la tasa de fallos como en AUC es muy

pequeña, por lo que no merece la pena aumentar aun más el tamaño de la muestra. Por el contrario, con tan solo 500 observaciones el error comienza a aumentar hasta el 11.5-12 %, además de disminuir el valor AUC ligeramente hasta 0.89; diferencias no muy significativas, pero que pueden solventarse aumentando a un tamaño moderado (1000).

Incluso si aumentamos el número de grupos y repeticiones a 10 y 20, respectivamente:



*Figure 43. Tasa de fallos y AUC, aumentando a 10 grupos y 20 repeticiones (Modelo 2)*

El orden de los modelos se mantiene idéntico. Por tanto, dado que el valor AUC es prácticamente el mismo, además de que la ganancia de la tasa de fallos es muy pequeña (dada la escala del eje), nos decantamos nuevamente por un modelo con *nodesize* = 20 y *sampszie* = 1000.

### RESUMEN *random forest*:

1. modelo 1: *mtry* = 3, *nodelsize* = 20 y *sampszie* = 1000.
2. modelo 2: *mtry* = 2, *nodelsize* = 20 y *sampszie* = 1000.

### 8.3 Comparación final

Por último, realizamos la comparación final de ambos modelos *random forest*:

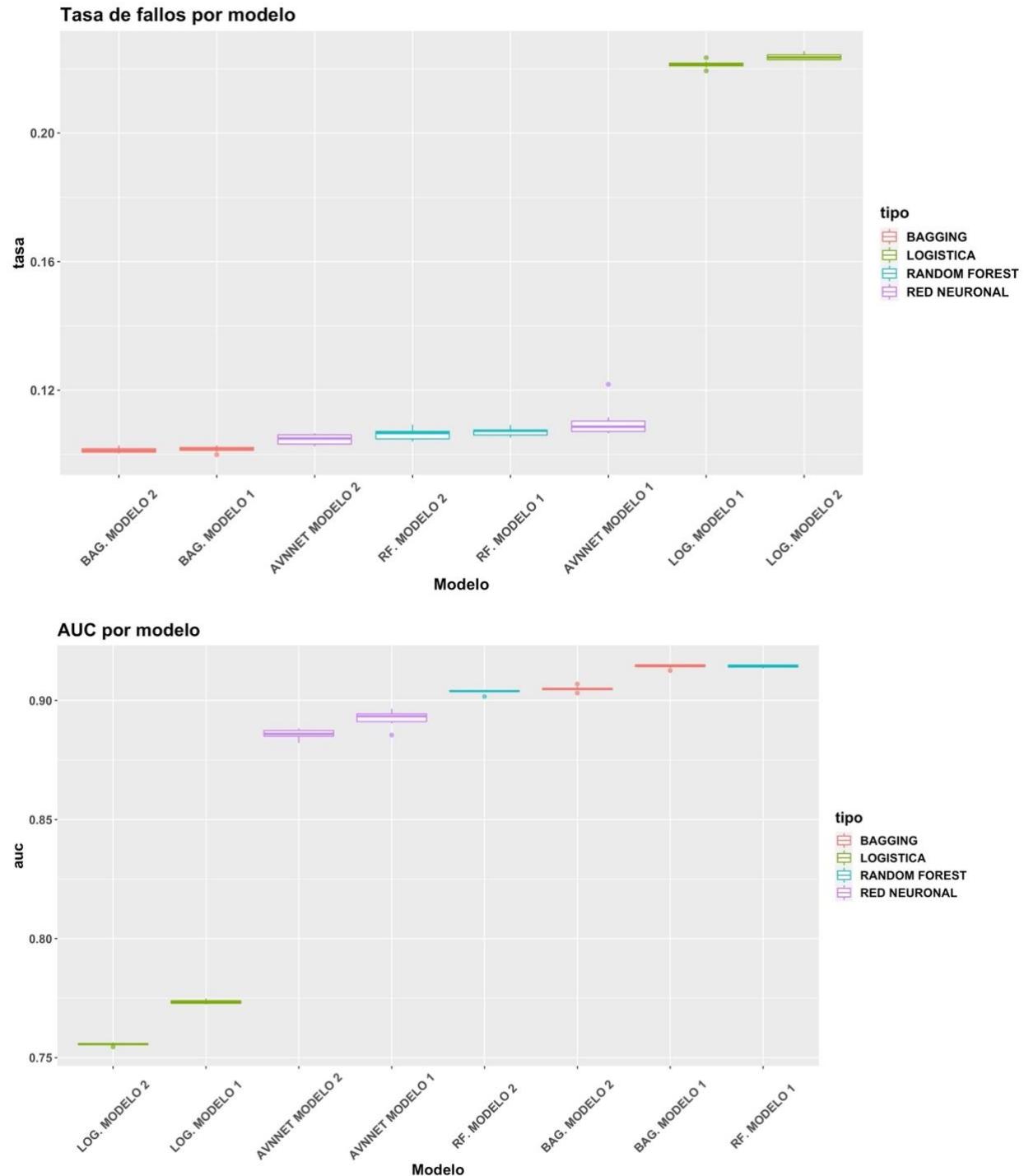


Figure 44. Comparación logística-avnnet-bagging-random forest

En relación con el resto de los modelos, *random forest* se sitúa a la misma altura que modelos como *bagging* o *avnnet*. De hecho, si hacemos "zoom" sobre los modelos de árbol:

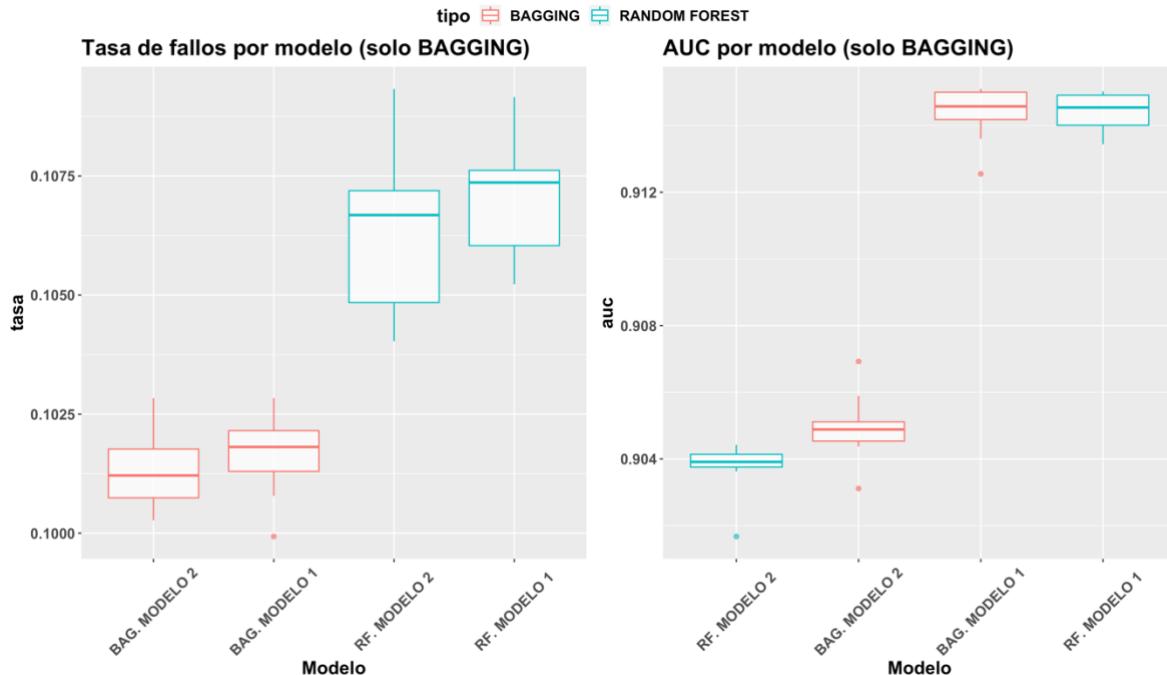


Figure 45. Comparación modelos bagging y random forest

Comprobamos que **la diferencia entre ambos no es muy significativa**, como es el caso de la tasa de fallos, donde la escala del eje puede llevar a engaño, ya que la diferencia entre ambos modelos es de apenas unas milésimas (de 0.1000 - 0.1025 en el caso de *bagging* a 0.1050-0.1075 en *random forest*). Además, aunque la varianza en este último parezca aumentar, dada la escala del eje no es tan pronunciada.

Por tanto, **no existe demasiada diferencia un modelo de árbol que sorteé todas las variables y un modelo en el que se utilicen todas**.

## 9. Gradient Boosting

Hasta el momento, hemos comprobado como la variación en el *accuracy*, tasa de error o AUC con los ambos *sets* candidatos no resulta ser relevante, incluso en numerosas ocasiones los parámetros finales acaban siendo los mismos para ambos candidatos. Por tanto, de cara al resto de modelos como *gradient boosting*, *xgboost* o *svm*, veamos si es posible poner en común un mismo tuneo de hiperparámetros para ambos *sets*, en lugar de "tunear" por separado cada uno de ellos.

### 9.1 Tuneo de hiperparámetros

En primer lugar, comenzamos recordando los parámetros a tunear en un modelo *gradient boosting*:

1. *shrinkage*: parámetro de regularización.
2. *n.minobsinnode*: tamaño máximo de nodos finales (parámetro encargado de medir la complejidad del modelo).

3. *n.trees*: número de iteraciones (árboles).
4. *bag.fraction*: fracción de observaciones del conjunto de entrenamiento seleccionadas aleatoriamente para la construcción del siguiente árbol.
5. *interaction.depth*: número de divisiones a realizar en cada rama del árbol (por defecto, lo establecemos a 2, esto es, árboles binarios):

Inicialmente, sobre ambos *sets* de variables realizamos un tuneo general sobre todos los hiperparámetros:

```
#-- Ejemplo de tuneo con el primer set de variables
set.seed(1234)
#-- El valor maximo de shrinkage suele estar en torno a 0.2
# (probamos con 0.3 y 0.4 tambien)
gbmgrid<-expand.grid(shrinkage=c(0.4,0.3,0.2,0.1,0.05,0.03,0.01,0.001),
                      n.minobsinnode=c(5,10,20),n.trees=c(100,500,1000,5000),
                      interaction.depth=c(2))

#-- De momento, mantenemos bag.fraction a 1 (todas las observaciones)
control<-trainControl(method="repeatedcv",number=5,savePredictions = "all",
                       repeats=5,classProbs=TRUE)
gbm_modelo1 <- train(formula_modelo_1,data=surgical_dataset,
                      method="gbm",trControl=control,tuneGrid=gbmgrid,
                      distribution="bernoulli", bag.fraction=1,verbose=FALSE)
```

## Modelo 1:

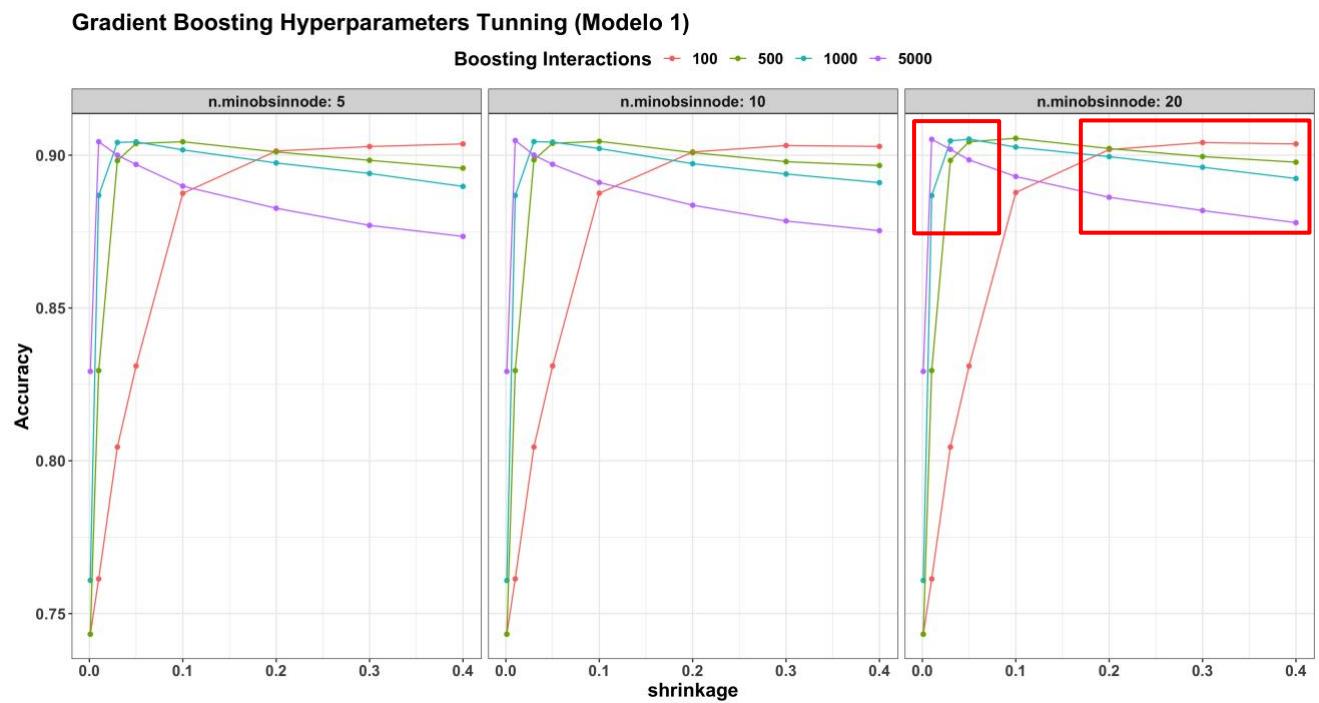


Figure 46. Tuneo hiperparámetros gbm (Modelo 1)

## Modelo 2:

Gradient Boosting Hyperparameters Tuning (Modelo 2)

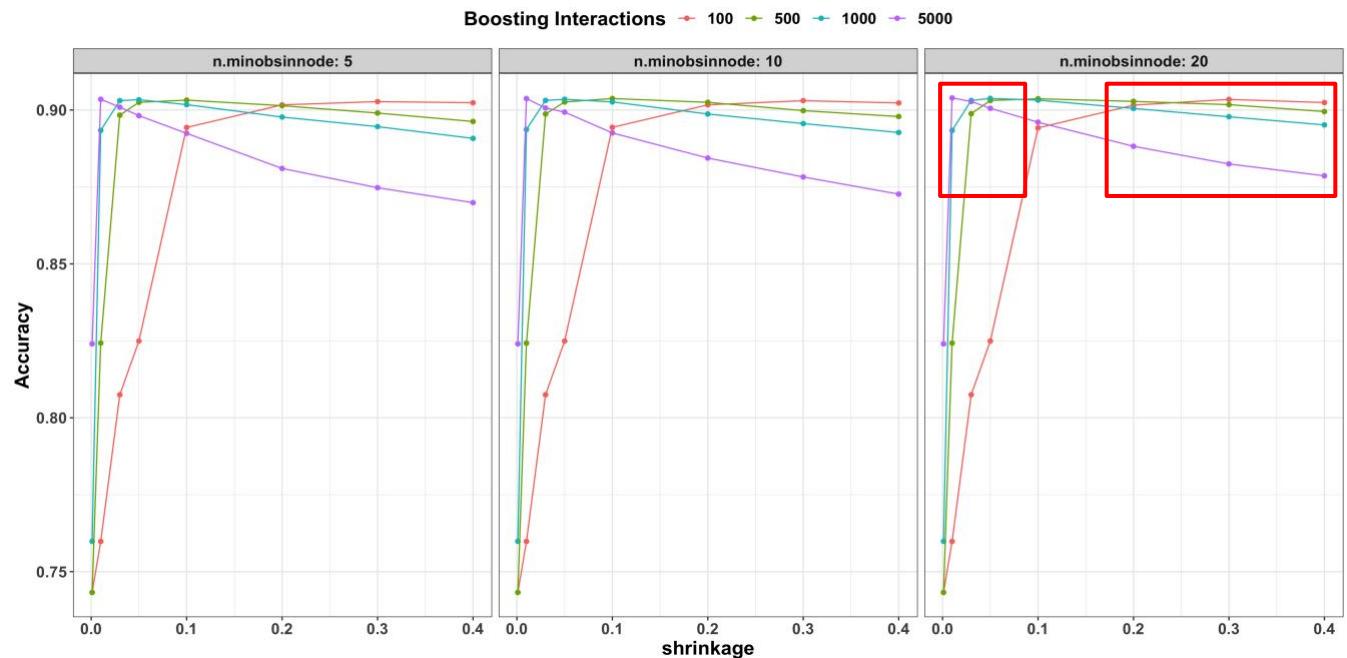


Figure 47. Tuneo hiperparámetros gbm (Modelo 2)

¿Qué sucede en ambos sets de variables? En primer lugar, **analicemos las recomendaciones de caret para ambos sets**:

Modelo 1: n.trees = 500, shrinkage = 0.1, n.minobsinnode = 20. Accuracy: 0.9055354  
 Modelo 2: n.trees = 5000, shrinkage = 0.01, n.minobsinnode = 20. Accuracy: 0.9039292

En función de lo recomendado por el propio paquete, existe un contraste entre ambos modelos: por un lado, **con cinco variables input recomienda 500 iteraciones, un valor de regularización alto y n.minobsinnode alto (20)**, es decir, **modelos más simples**. Por el contrario, **con una variable input menos, recomienda muchos más árboles (5000) y un parámetro de regularización más bajo**.

Desde un punto de vista numérico, con estos parámetros se han obtenido el *accuracy* máximo (incluso por la estructura de remuestreo, el valor de la semilla empleada, etc.). Sin embargo, ¿Es una configuración óptima? O aún más importante ¿Podrían obtenerse resultados similares con modelos aún más sencillos? Para responder a ambas preguntas, **debemos analizar los posibles patrones que presenta el modelo en ambos sets**, de forma que podamos decantarnos por una opción mucho más sencilla.

Concretamente, destacamos tres grandes patrones de comportamiento:

1. Por un lado, **un bajo número de iteraciones (en torno a 100), junto con un valor de regularización alto (shrinkage superior o igual a 0.2-0.3, aproximadamente)**.
2. Por el contrario, también es posible obtener un máximo *accuracy* **con un valor shrinkage bajo (inferior a 0.1), pero con un elevado número de iteraciones, en torno a 500, 1000 o 5000**.

- Además, el comportamiento de  $n.minobsinnode$  en ambos *sets* es el mismo tanto para un valor de 5, como de 10 o 20, **por lo que un valor alto (20) es una buena opción por la que decantarse: a un mayor valor  $n.minosinnode$ , obtenemos modelos más simples y con menos posibilidad de sobreajuste.**

En conclusión, **no es necesario tunear exageradamente el modelo en ambos casos**, sino que aparentemente con un número de árboles bajo (100), un valor de regularización alto (en torno a 0.2-0.3) y  $n.minobsinnode = 20$  se obtienen de por sí buenos resultados, sin necesidad de complicar aún más el modelo.

### 9.1.1 Estudio del *Early Stopping*

Continuando con la línea anterior, para ambos *sets* de variables **probamos a fijar  $n.minobsinnode = 20$  y  $shrinkage = 0.2$  y 0.3**, variando el número de iteraciones. De este modo, se pretende no solo observar qué número de iteraciones/árboles sería adecuado utilizar, sino además si con un valor de regularización de 0.2 es suficiente o si aumentando su valor a 0.3 obtenemos mejores resultados, ya que en el gráfico anterior se detectó un ligero ascenso en el valor de *accuracy* en 0.3 (normalmente el valor máximo suele ser 0.2):

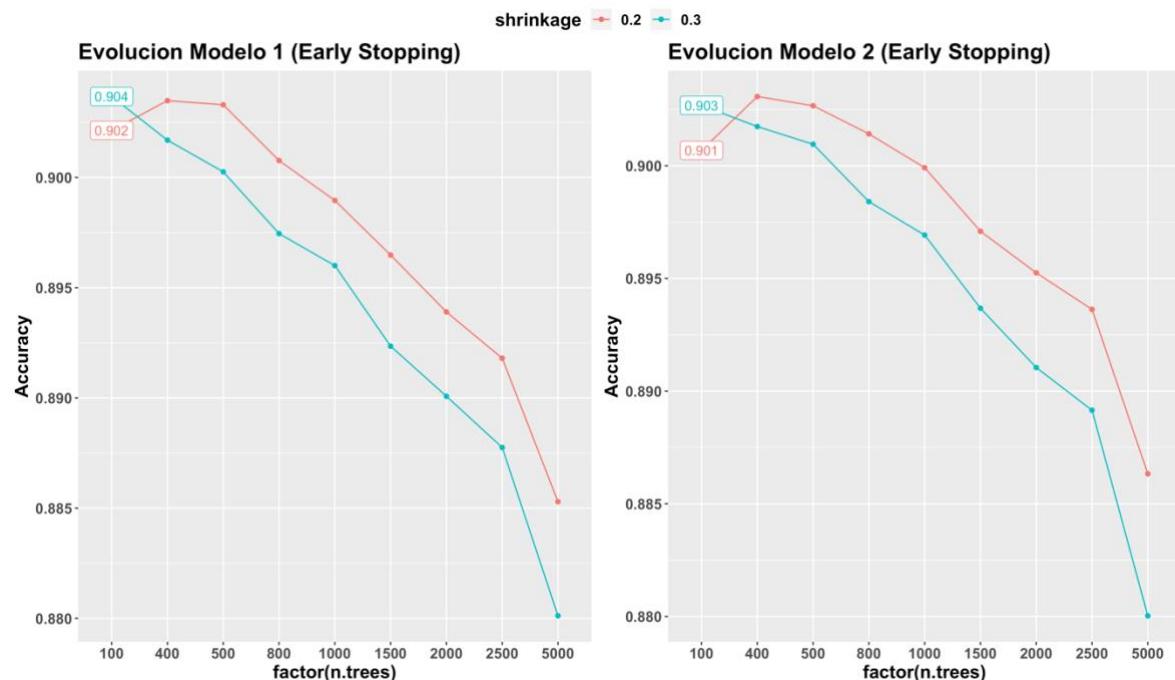


Figure 48. Estudio Early Stopping (Modelos 1 y 2)

En ambos *sets*, **observamos que con 100 árboles se obtiene un buen accuracy**, mientras que a partir de 400-500 iteraciones, el modelo tiende al sobreajustarse y, como consecuencia, disminuye su precisión.

Por otro lado, en relación con el parámetro *shrinkage* no existe apenas diferencia entre un valor 0.2 y 0.3, por lo que **nos decantamos por 0.2**.

### 9.1.2 Estudio de *bag.fraction*

Hasta el momento, hemos tuneado los modelos *gbm* con el conjunto total de las observaciones, esto es, con *bag.fraction* = 1. Sin embargo, por defecto *caret* no sorteá todas las observaciones, sino que por defecto escoge el 50 % aleatoriamente (0.5)<sup>6</sup>

Como consecuencia, sobre ambos *sets* estudiamos la variabilidad del modelo con respecto a dicho parámetro, aumentando a 10 el número de repeticiones para observar mejor su efecto. Dado que la función *cruzadagbm* no dispone de dicho parámetro, se ha incluido internamente en la función proporcionada por el profesor:

```
##-- Incluimos bag.fraction dentro de train (en cruzadagmbin.R)
gbm <- train(formula, data=databis, method="gbm", trControl=control,
              bag.fraction=bag.fraction, tuneGrid=gbmgrid, distribution="bernoulli")
```

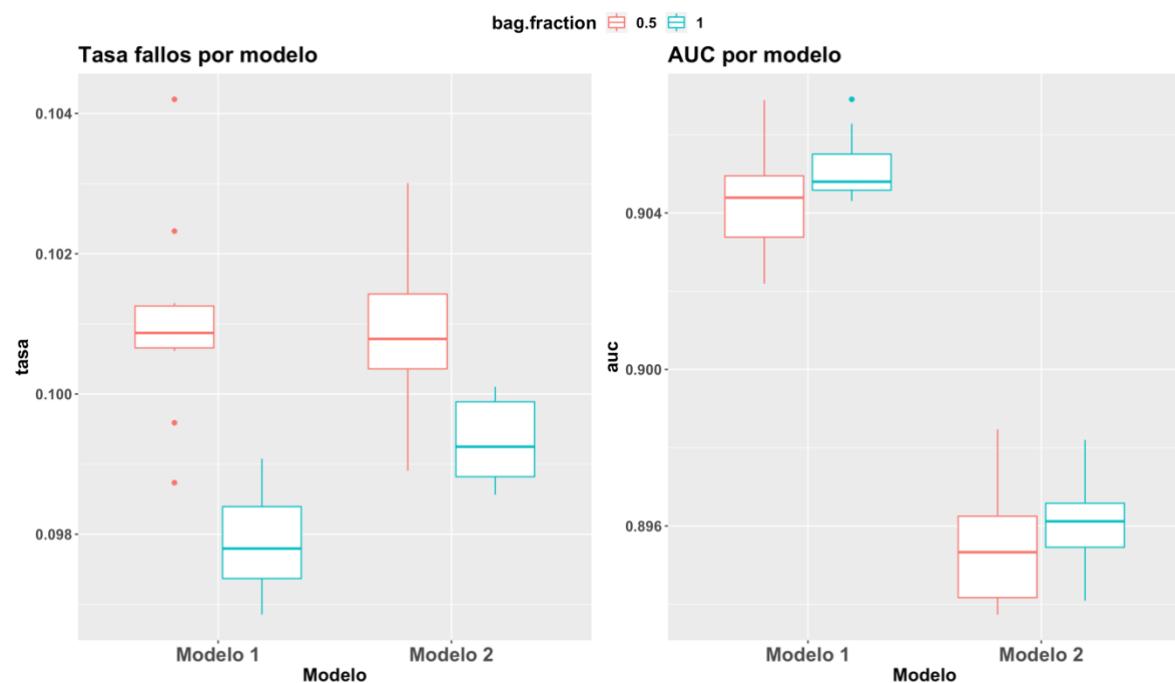


Figure 49. Estudio *bag.fraction* modelos 1 y 2 (con 10 repeticiones)

En un primer análisis, la diferencia entre *bag.fraction* = 0.5 y *bag.fraction* = 1 es, en ambos *sets*, poco relevante, ya que tanto en sesgo como en varianza presentan resultados muy similares, salvo en la tasa de fallos donde aumenta ligeramente, aunque la dada la escala de los ejes no es tan significativa. A modo de ejemplo, en el modelo 1 la varianza de la tasa de fallos se sitúa entre 0.099 y 0.104, considerando los *outliers*; y en el segundo modelo apenas también aumenta su varianza en unas milésimas (de 0.099 a 0.103).

Por tanto, no existe apenas diferencia entre utilizar el 100 % de las observaciones y tan solo el 50 %, por lo que podemos mantener dicho parámetro por defecto en *bag.fraction*.

---

<sup>6</sup> *caret* emplea internamente el paquete *gbm*, cuyo valor *bag.fraction* por defecto es 0.5. <https://cran.r-project.org/web/packages/gbm/gbm.pdf>

## RESUMEN *gradient boosting*:

- modelos 1,2:  $n.trees = 100$ ,  $shrinkage = 0.2$ ,  $n.minobsinnode = 20$  y  $bag.fraction = 0.5$ .

## 9.2 Comparación final

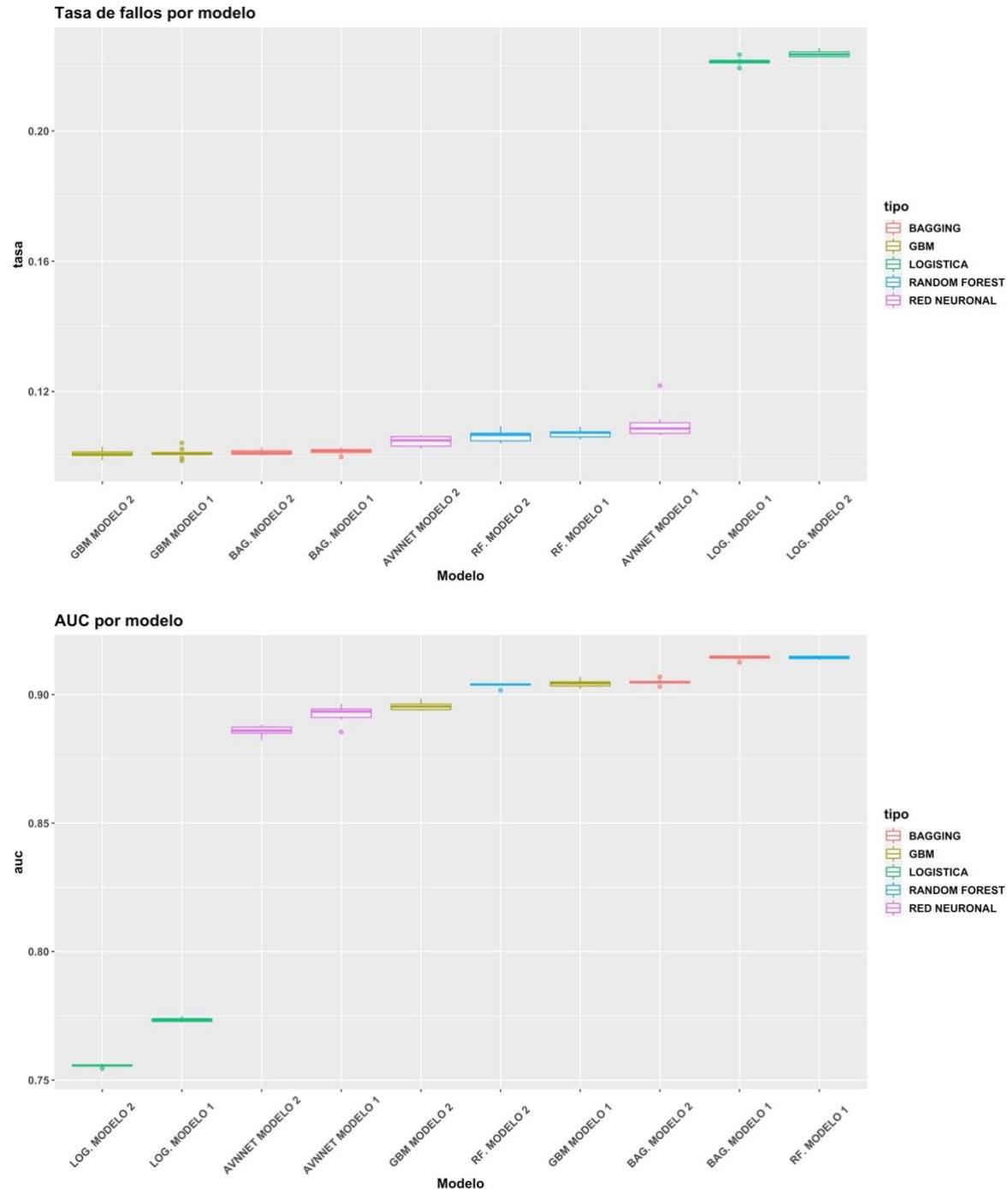


Figure 50. Comparación logística-avnnet-bagging-random forest-gbm

Como podemos comprobar, *gradient boosting* se sitúa prácticamente en la misma línea que el resto de los modelos basados en árbol. De hecho, aunque la tasa de fallos en *gradient boosting* sea ligeramente menor, **modelos como random forest o bagging continúan siendo una buena alternativa**, dado que obtienen modelos similares con menos muestras (con 1000 observaciones en lugar del 50 % de los datos, además de sortear variables).

Por otro lado, incluso con *gbm* la diferencia entre ambos *sets* candidatos resulta insignificante (de 0.899 a 0.905 en el caso del AUC), por lo que el modelo 2, con tan solo cuatro variables, continúa siendo la mejor opción.

## 10. Support Vector Machines

### 10.1 SVM Lineal

A continuación, continuamos con los modelos *svm*, comenzando por el *kernel* más sencillo: el lineal, del cual solo necesitamos tunear el parámetro de regularización **C** (por el momento, empleamos 5 repeticiones):

```
##-- Probamos desde C = 0.01 hasta C = 10
C_binaria <- expand.grid(C=c(0.01,0.05,0.1,0.2,0.5,1,2,5,10))
##-- Ejemplo con el primer set de variables (5 variables input)
cruzadaSVMbin(data=surgical_dataset, vardep=target,
               listconti = var_modelo1, listclass=c(""),
               grupos=5,sinicio=1234,repe=5,C=C_binaria)
```

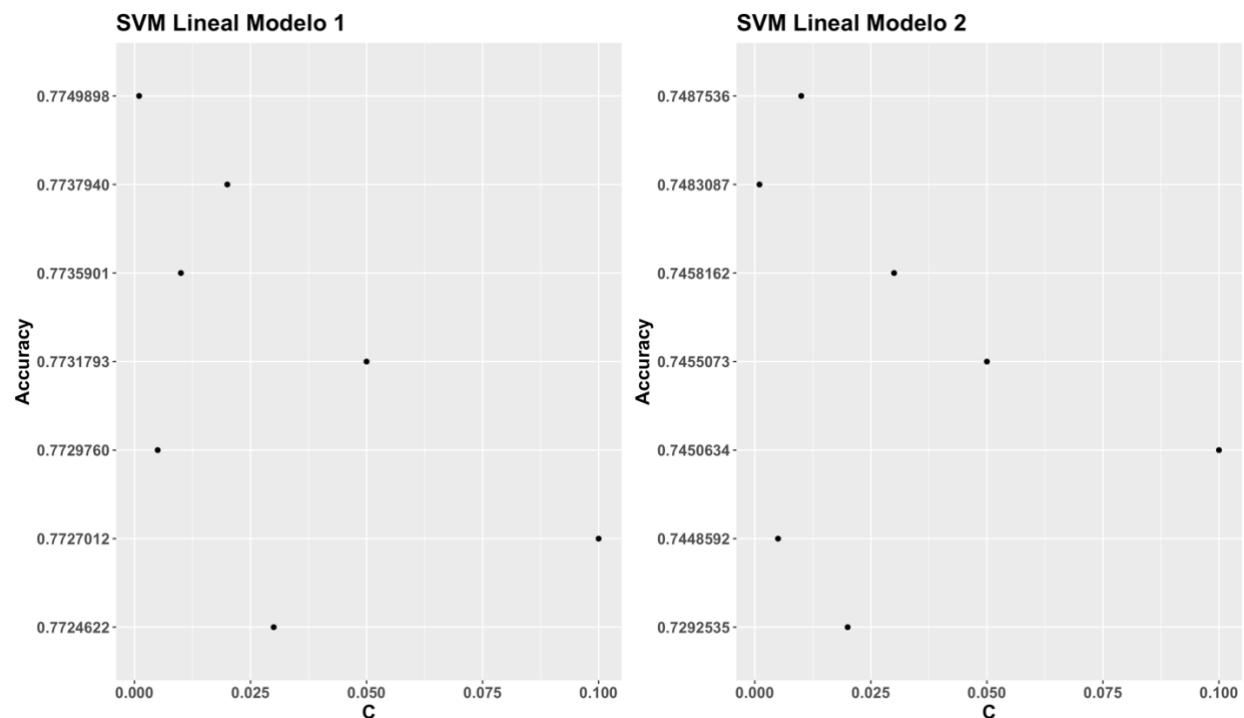


Figure 51. Distribución SVM Lineal Modelos 1 y 2

Recomendado por caret

Modelo 1: C = 0.01. Accuracy: 0.7746143

Modelo 2: C = 0.01. Accuracy: 0.7466688

A diferencia del resto de modelos no lineales, con el primer *set* de variables (5 variables *input* incluyendo *ccsMort30Rate*), obtenemos un *accuracy* en torno al 77 %, mientras que con tan solo 4 variables desciende al 74 %, un indicativo de la separación lineal del ratio de mortalidad (*ccsMort30Rate*) sobre la variable objetivo. Pese a ello, los modelos basados en árbol o *boosting* continúan siendo mejores en términos de *accuracy* (89-90 %).

En relación con la constante C, dada la escala de los ejes **no se aprecia una diferencia relevante entre los diferentes valores**, ya que apenas varían en unas milésimas. Sin embargo, si se tuviera que indicar un posible patrón, diríamos que **a medida que se reduce la constante C, aumenta ligeramente el accuracy**. Podríamos incluso probar a reducir aún más el parámetro de regularización:

```
#-- Probamos a reducir C entre 0.1 y 0.001  
C_binaria <- expand.grid(C=c(0.001,0.005,0.01))
```

Pero tan solo aumenta en 0.0001 en ambos *sets*:

```
Modelo 1: C: 0.01 -> 0.7746143 ; C: 0.001 -> 0.7749898  
Modelo 2: C: 0.01 -> 0.7466688 ; C: 0.001 -> 0.7749898
```

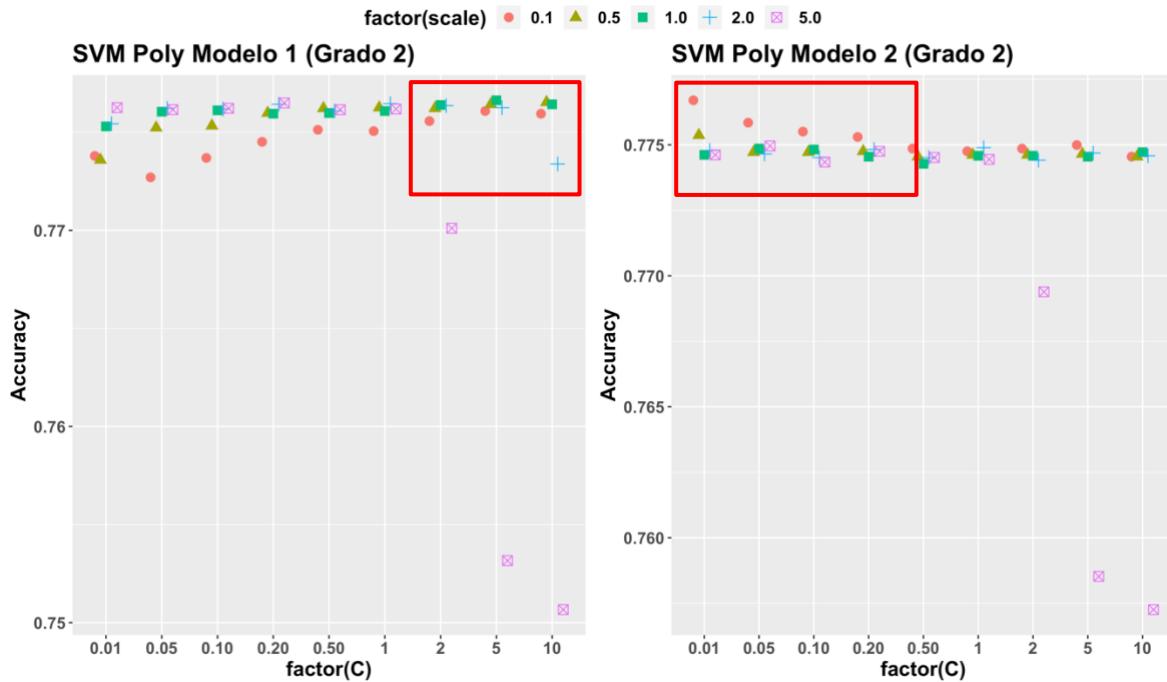
Por tanto, y dado que la diferencia no es muy significativa, **nos decantamos por el máximo accuracy, correspondiente con C = 0.01**, recomendado por *caret*, un valor no demasiado alto, lo que se traduce en un menor riesgo por sobreajuste (mayores márgenes de separación).

## 10.2 SVM Polinomial

A continuación, proseguimos con el *kernel* polinomial, **tuneando tanto la constante C de regularización como la escala** (nuevamente, con cinco repeticiones):

**NOTA:** dado que los polinomios de grado 3 tardaban demasiado tiempo en procesar (incluso con una sola repetición + *doParallel*), se ha utilizado únicamente grado 2.

```
#-- Probamos con C desde 0.01 hasta 10  
C_poly      <- c(0.01, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10)  
degree_poly <- c(2)  
scale_poly   <- c(0.1, 0.5, 1, 2, 5)  
#-- Ejemplo con el primer set de variables (5 variables input)  
svm_pol_1 <- cruzadaSVMbinPoly(data=surgical_dataset, vardep=target,  
                                listconti = var_modelo1, listclass=c(""),  
                                grupos=5,sinicio=1234,repe=5,C = C_poly,  
                                degree = degree_poly, scale = scale_poly)
```



*Figure 52. Distribución SVM Polinomial Modelos 1 y 2 (I)*

Recomendado por caret

Modelo 1:  $C = 5$ ; degree = 2; scale = 0.1. Accuracy: 0.7766301

Modelo 2:  $C = 0.01$ ; degree = 2; scale = 0.1. Accuracy: 0.7766992

En base a la respuesta del paquete *caret*, con el primer *set* recomienda una escala en torno a 0.1 y una constante C alta (5), mientras que en el caso del segundo *set* de variables, la constante C recomendada es más baja, en torno a 0.01. No obstante, y dado que **no podemos dejarnos llevar por valores puntuales (pues puede conducir al sobreajuste)**, analicemos los patrones existentes:

1. En el caso del primer *set* de variables, nos encontramos con **una constante C alta (entre 5 y 10) y una escala moderada/pequeña (en torno a 0.1-0.5-1)**, **patrón que aumenta conforme se incrementa la constante C**. No obstante, el aumento no parece ser muy significativo dada la escala de los ejes. A modo de ejemplo, con  $C = 0.5$ , el *accuracy* con un *scale* bajo se sitúa en torno a 0.775, aproximadamente. Sin embargo, al aumentar  $C = 10$ , el *accuracy* tan solo aumenta hasta 0.777.
2. En el segundo *set*, por otro lado, **llama la atención modelos con una constante C pequeña (menor a 0.01) y una escala baja (0.1-0.5, principalmente)**. Es decir, a medida que disminuye C (con un valor *scale* pequeño), aumenta el valor de *accuracy*, aunque de nuevo, dada la escala de los ejes, el aumento no es muy relevante.

Pese a ello, analicemos si mejoran los resultados en caso de, en el primer *set*, aumentar la constante C a 15 y 20; además de probar a reducir C a 0.005 y 0.001 en el segundo *set* de variables:

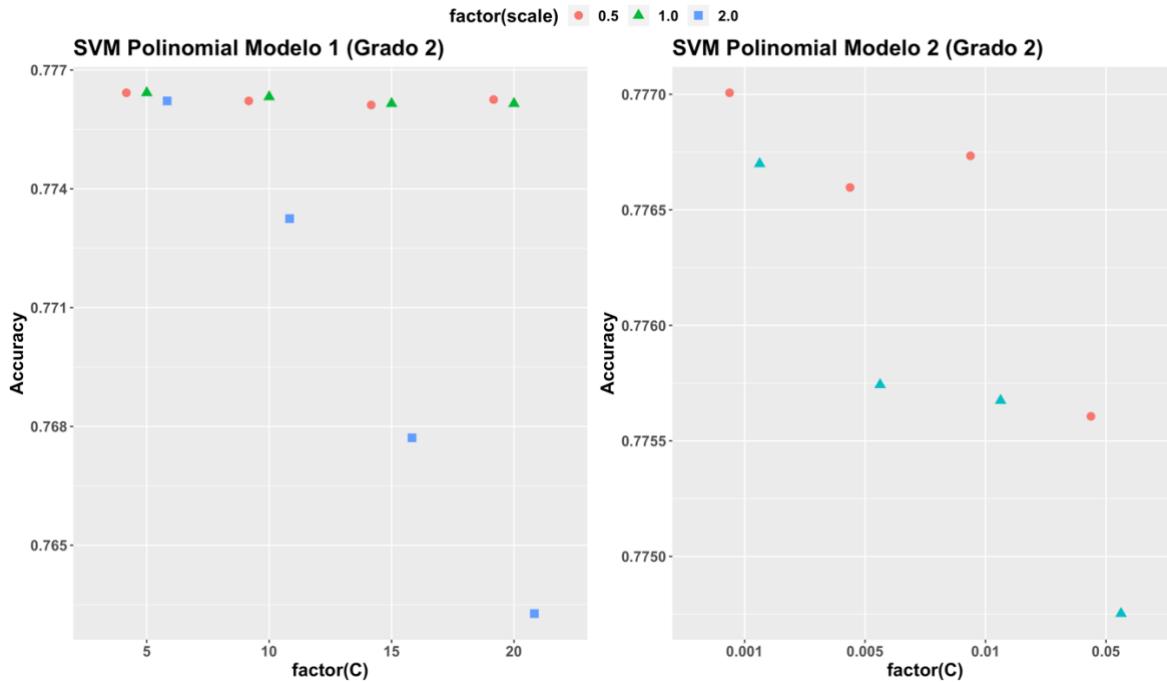


Figure 53. Distribución SVM Polinomial Modelos 1 y 2 (II)

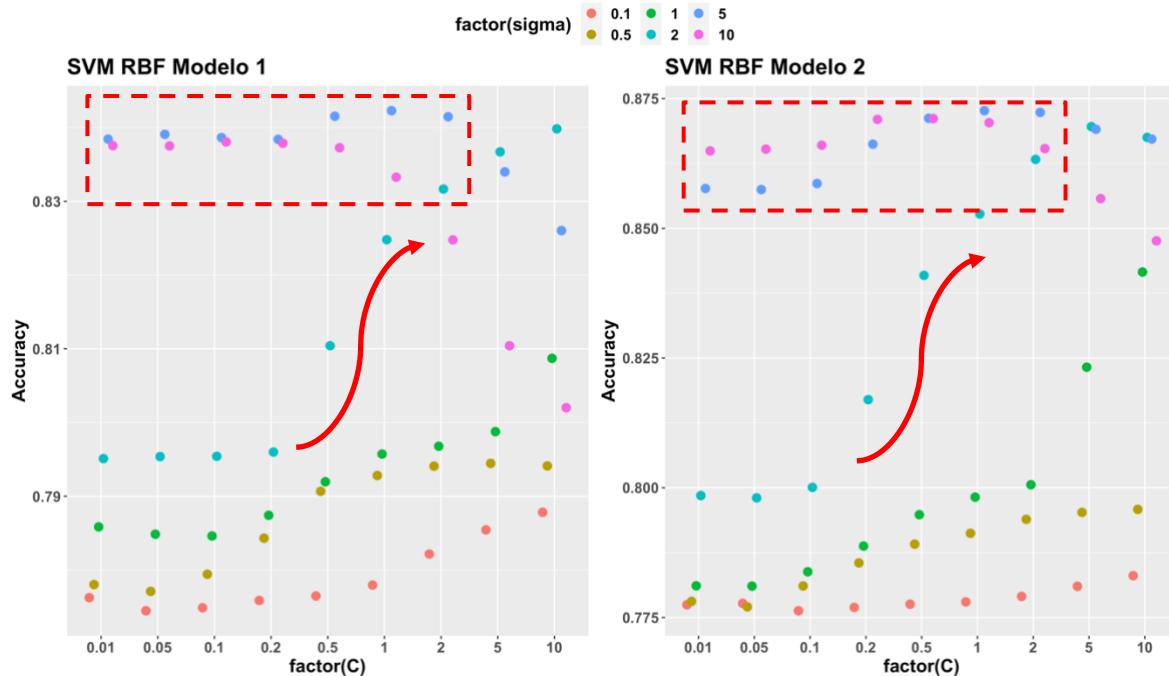
1. En el caso del primer set, pese a aumentar la constante C, el valor de *accuracy* se mantiene en torno a 0.77 con un *scale* bajo. Por tanto, y dado que la diferencia no es muy significativa entre un valor C alto y pequeño, **podemos decantarnos por un valor en torno a 0.5 (moderado-bajo), así como una escala de 1**, aunque *caret* nos recomienda un valor C = 5, de forma que los márgenes de separación entre ambas clases son mayores, obteniendo un modelo más "general".
2. Por otro lado, en relación con el segundo set, habiendo disminuido el valor C hasta 0.001-0.005, **no ha hecho aumentar en gran medida el resultado**. Por tanto, dado que con un valor C pequeño no solo obtenemos un mayor margen de separación o maniobrabilidad, sino además el mismo *accuracy* que el de un modelo más "estricto", **nos decantamos por la opción de caret**: C pequeño (0.01) y escala pequeña (0.1).

Cabe recalcar que **las decisiones están basadas por patrones y no en valores puntuales**, por lo que empleando otra semilla y con otros valores training los parámetros serán "aproximadamente" similares, aunque no iguales.

### 10.3 SVM RBF

Finalizando con el *kernel* radial, comenzamos tuneando los dos parámetros principales, C y  $\sigma$ :

```
#-- Probamos con C entre 0.01 y 10
C_rbf      <- c(0.01, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10)
sigma_rbf   <- c(0.1, 0.5, 1, 2, 5, 10)
#-- Ejemplo con el primer set variables
svm_rbf_1 <- cruzadaSVMbinRBF(data=surgical_dataset, vardep=target,
                                listcont = var_modelo1, listclass=c(""),
                                grupos=5, sinicio=1234, repe=5, C = C_rbf,
                                sigma = sigma_rbf)
```



*Figure 54. Distribución SVM RBF Modelos 1 y 2 (I)*

Recomendado por caret

Modelo 1: C = 1 ; sigma = 5 ; Accuracy: 0.8422943

Modelo 2: C = 1 ; sigma = 5 ; Accuracy: 0.8726666

En base a los resultados devueltos por *caret*, para ambos *sets* de variables recomienda una constante C pequeña/moderada (1), así como un valor  $\sigma = 5$ . No obstante, y dado que se tratan de valores puntuales, analicemos los patrones existentes:

1. En el caso del primer *set*, los modelos con mayor *accuracy* presentan un valor *sigma* alto (en torno a 5-10 aumenta hasta 0.83). En relación con la constante C, con un valor bajo (y sigma alto) el *accuracy* se sitúa en torno a 0.86-0.87, alcanzando su punto máximo en C = 1. Es decir, no es necesario un valor C alto (márgenes de separación más pequeños), ya que a partir de C = 2-5, el *accuracy* comienza a disminuir. No obstante, otro posible patrón de comportamiento sería con un valor *sigma* pequeño (2) y C alto (a partir de 5-10), donde comienza a aumentar el *accuracy*, al mismo ritmo que desciende con sigma = 5 o 10.
2. En el caso del segundo modelo, el comportamiento es bastante similar: o bien escogemos valores *sigma* altos (5-10) y C no demasiado alto ( $\leq 1$ ), o bien un valor *sigma* menor (2) y una constante C alta (en torno a 5, 10).

En ambos *sets*, podemos comprobar cómo evoluciona el modelo si aumentamos el valor de C con *sigma* = 2, dado que a simple vista comienza a aumentar conforme incrementamos C de forma progresiva (de 0.775 a 0.875):

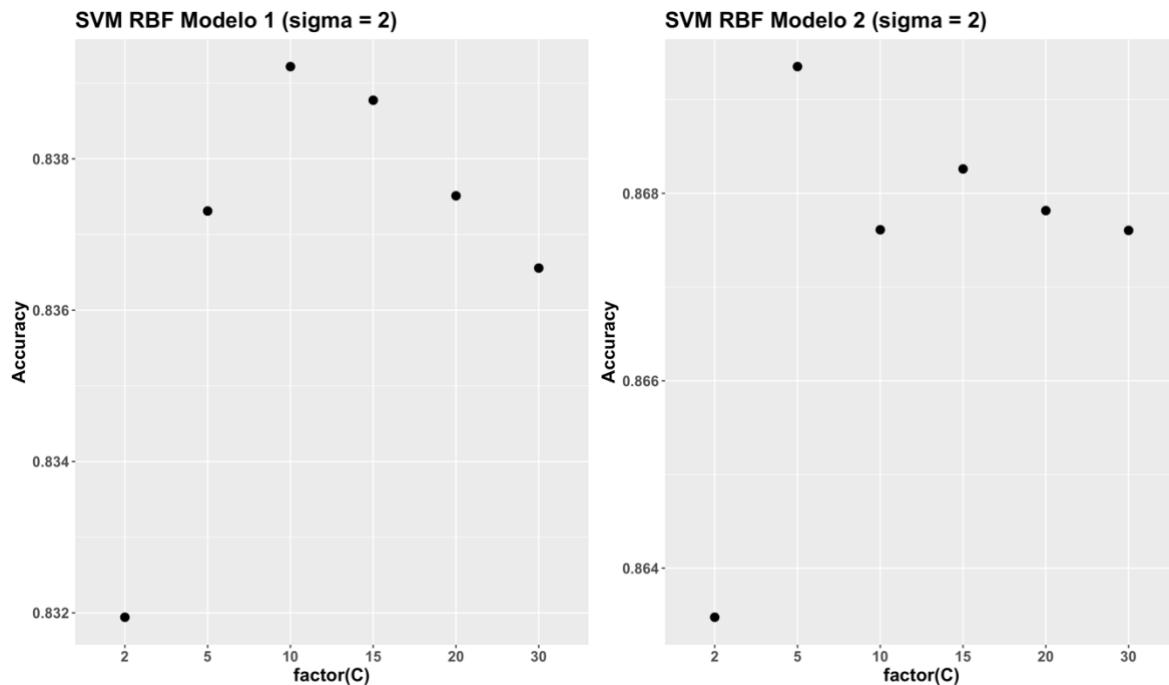


Figure 55. Distribución SVM RBF Modelos 1 y 2 (II)

Incluso aumentando  $C$ , se obtienen prácticamente los mismos resultados que con un *sigma* y  $C$  menor. En conclusión, para ambos *sets* de variables **nos decantamos por un valor *sigma* alto (escogemos 5), y una constante  $C$  no demasiado alta, entre 0.5 y 1:**

Recomendado por caret

Modelo 1:  $C = 1$  ;  $\sigma = 5$  ; Accuracy: 0.8422943  
 $C = 0.5$  ;  $\sigma = 5$  ; Accuracy: 0.8415420

Modelo 2:  $C = 1$  ;  $\sigma = 5$  ; Accuracy: 0.8726666  
 $C = 0.5$  ;  $\sigma = 5$  ; Accuracy: 0.8711983

Dado que la diferencia entre  $C = 1$  y  $C = 0.5$  no es muy relevante (aunque caret recomienda 1), resulta indiferente escoger uno de ellos (por ejemplo, escogemos 0.5).

### RESUMEN SVM Lineal:

- modelos 1 y 2:  $C = 0.01$ .

### RESUMEN SVM Polinomial:

- modelo 1:  $C = 0.01$  ;  $scale = 0.1$ .
- modelo 2:  $C = 0.5$  ;  $scale = 1$ .

### RESUMEN SVM RBF:

- modelos 1 y 2:  $C = 0.5$  ;  $\sigma = 5$ .

## 10.4 Comparación modelos SVM

Una vez elaborados los modelos, empleando validación cruzada repetida de 5 grupos y 10 repeticiones, analizamos tanto la tasa de fallos como el AUC (en ambos *sets* de variables):

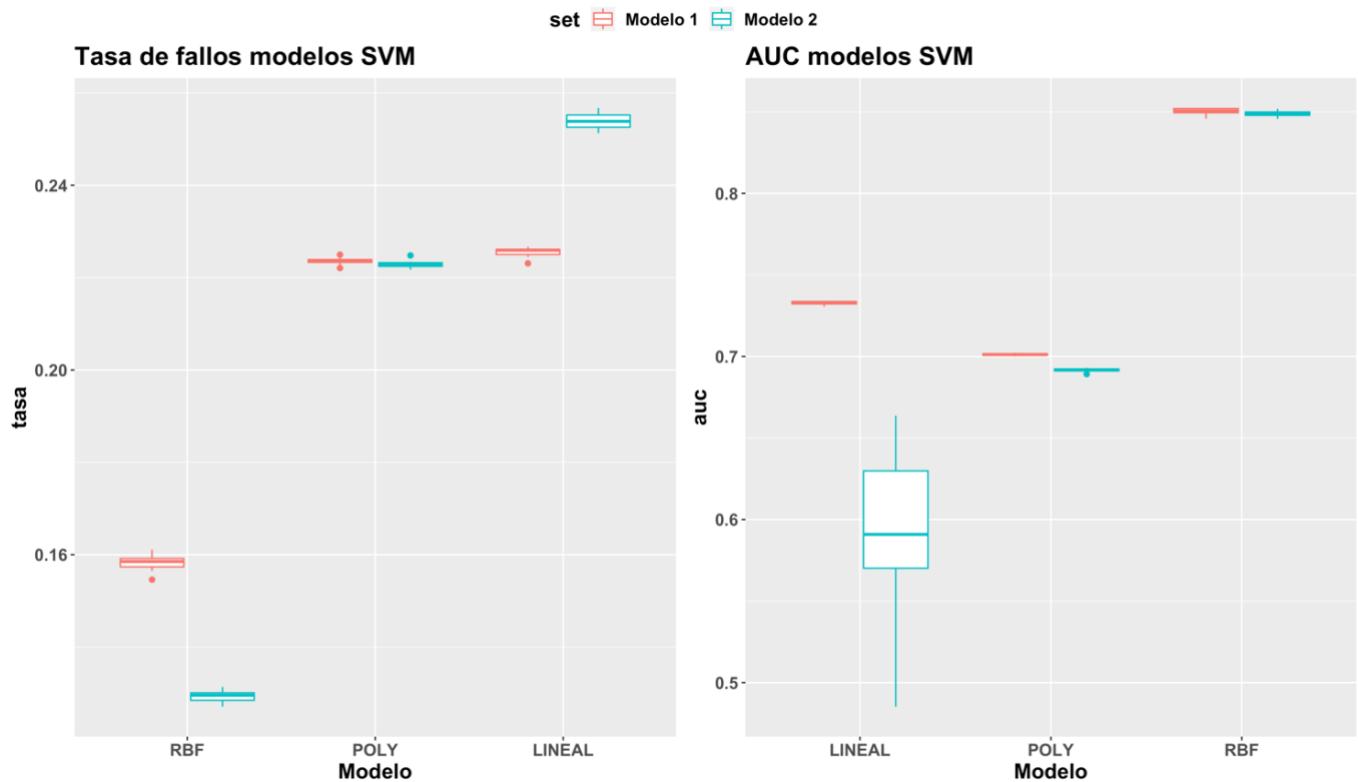


Figure 56. Comparación tasa de fallos y AUC (modelos SVM)

Sin duda alguna, la mejor alternativa tanto en tasa de fallos como en AUC es el modelo SVM con *kernel* radial (RBF). De hecho, debemos remarcar dos detalles fundamentales, relacionados con ambos *sets* de variables:

1. Por un lado, en relación con el *kernel* lineal observamos una considerable diferencia entre el primer *set* con cinco variables (con *ccsMort30Rate*) y el segundo *set*, tanto en AUC, sesgo y varianza: incluyendo dicha variable (mayor separación lineal sobre *target*), el modelo mejora hasta 0.74-0.75 en AUC, aproximadamente, además de reducir la tasa de error hasta 0.22 (a diferencia del primer *set*, que alcanza 0.26), y una varianza en las "cajas" significativamente menor; mientras que con 4 variables el modelo resulta más impredecible.
2. Por el contrario, con el *kernel* radial, añadir dicha variable perjudica al modelo en cuanto a la tasa de fallos (de 0.16 con cinco variables a 0.12-0.13 con tan solo cuatro), aunque en AUC se mantienen prácticamente idénticos.

A la vista de los resultados obtenidos, los modelos SVM RBF resultan ser claramente los ganadores, con tasas de fallos y AUC similares al de un modelo de red, pero inferiores (ligeramente) con respecto a modelos basados en árbol o gradient boosting:

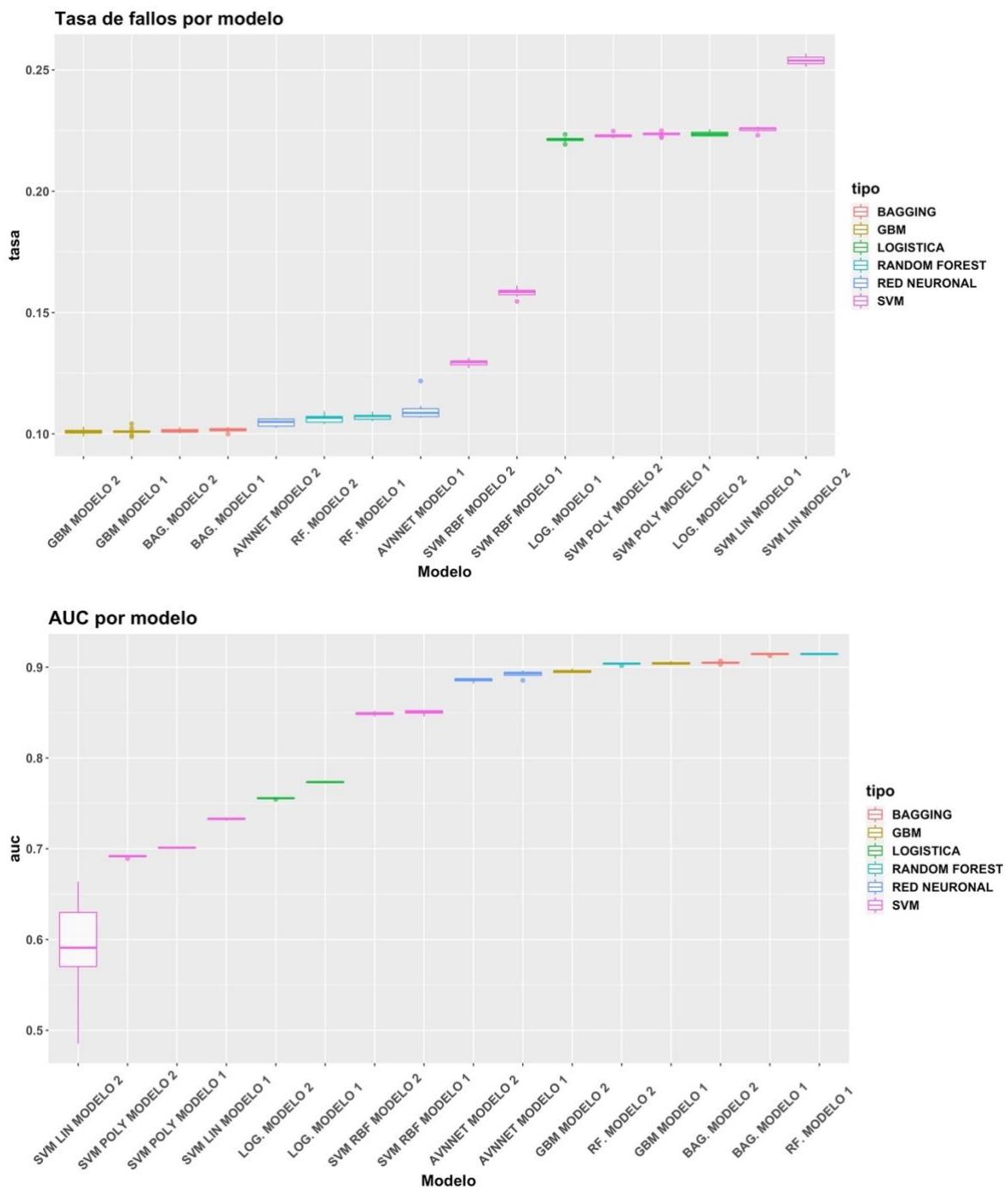


Figure 57. Comparación logística-avnnet-bagging-randomforest-gbm-svm

Antes de continuar con *XGboost* y Ensamblado, debemos detenernos a analizar **si realmente es necesario continuar con ambos sets de variables o no**. A lo largo de la práctica, se ha podido comprobar que la diferencia entre ambos candidatos, por tan solo una variable *input*, no ha sido especialmente relevante, pese a lo observado en la regresión logística donde si existía cierta diferencia entre ambos.

Por tanto, de cara a *XGboost* pero especialmente a los modelos de Ensamblado **continuaremos únicamente con el segundo set de variables input (4 variables)**, a excepción del modelo **svm** lineal, donde utilizaremos el modelo obtenido con el primer set, con el que se ha conseguido mejores resultados.

## 11. XGboost

### 11.1 Tuneo de hiperparámetros

En primer lugar, debemos recordar los parámetros a tunear en un modelo *XGboost*:

1. *min\_child\_weight*: número de observaciones mínimas en el nodo final. Similar a *n.minobsinnode* de *gbm*.
2. *eta*: parámetro de regularización.
3. *rounds*: número de iteraciones.
4. *max\_depth*: profundidad máxima de los árboles.
5. *gamma*: constante de regularización (lo mantenemos a 0).
6. *subsample*: sorteo de observaciones antes de cada árbol (lo mantenemos a 1).

```
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=6,gamma=0,
                           colsample_bytree=1,subsample=1)
```

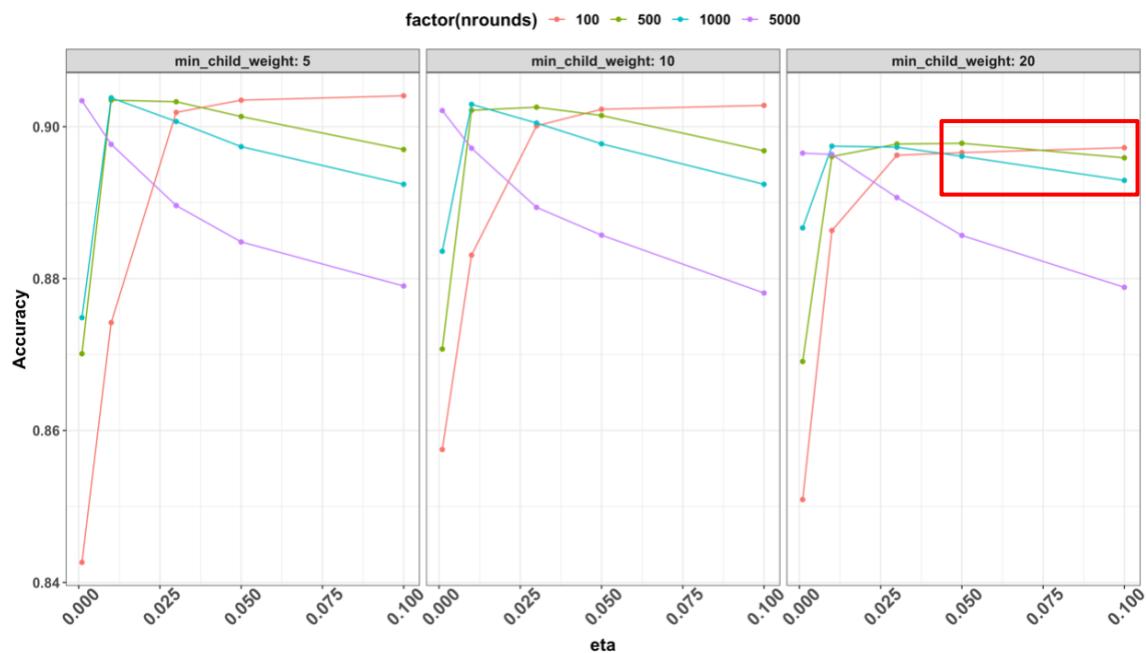
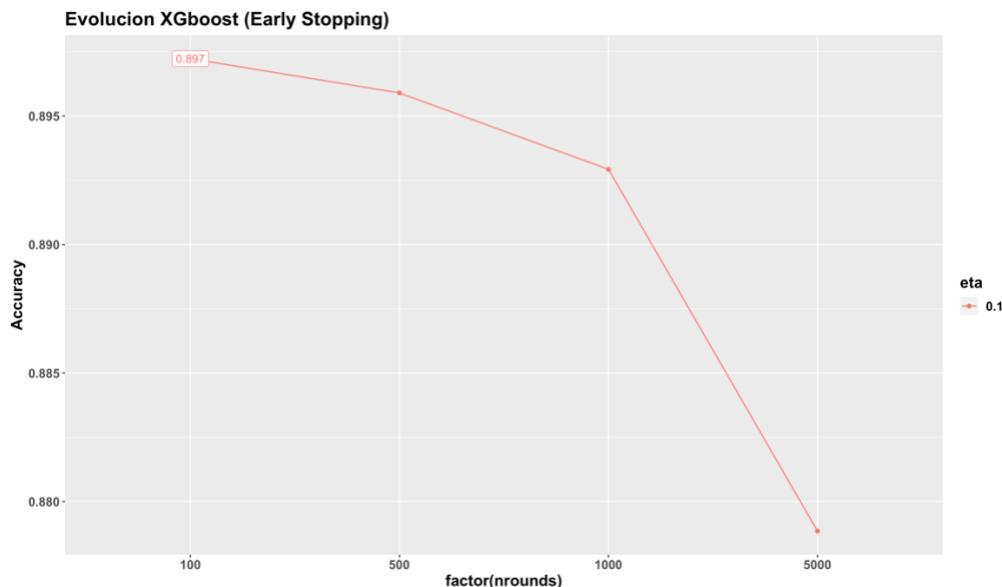


Figure 58. Tuneo hiperparámetros XGboost (5 grupos y 5 repeticiones)

De los gráficos anteriores, se deduce que se debe tomar un valor *eta* alto (en torno a 0.1) y pocas iteraciones, alrededor de 100. Además, si nos fijamos en la escala de los ejes, no existe demasiada diferencia entre un valor *min\_child\_weight* bajo (5) y alto (20), tan solo de apenas unas décimas (0.90 y 0.89, respectivamente). Por tanto, escogemos un valor de 20 al ser las diferencias tan pequeñas, permitiendo con ello modelos más simples y con menos sobreajuste.

## 11.2 Estudio *Early Stopping*

A continuación, probamos a fijar algunos parámetros para analizar cómo evoluciona el modelo en función del número de iteraciones:



*Figure 59. Evolución Early Stopping XGboost*

Como podemos comprobar en el gráfico anterior, **100 iteraciones es un valor óptimo**, en términos generales.

## 11.3 Tuneo *max\_depth*

Hasta el momento, se ha considerado una profundidad máxima de 6 en cada árbol del modelo. No obstante, **analicemos la evolución del modelo, tanto en sesgo como en varianza, al cambiar dicho parámetro**:

```
#-- Tuneamos max_depth (minimo 1, maximo 20)
xgbm_mo <- cruzadaxgbmbin(data=surgical_dataset, vardep=target,
                           listconti=var_modelo2, listclass=c(""),
                           grupos=5, sinicio=1234, repe=5,
                           nrounds=100, eta=0.01, min_child_weight=20,
                           gamma=0, colsample_bytree=1, subsample=1,
                           max_depth=c(1,3,6,10,15,20))
)
```

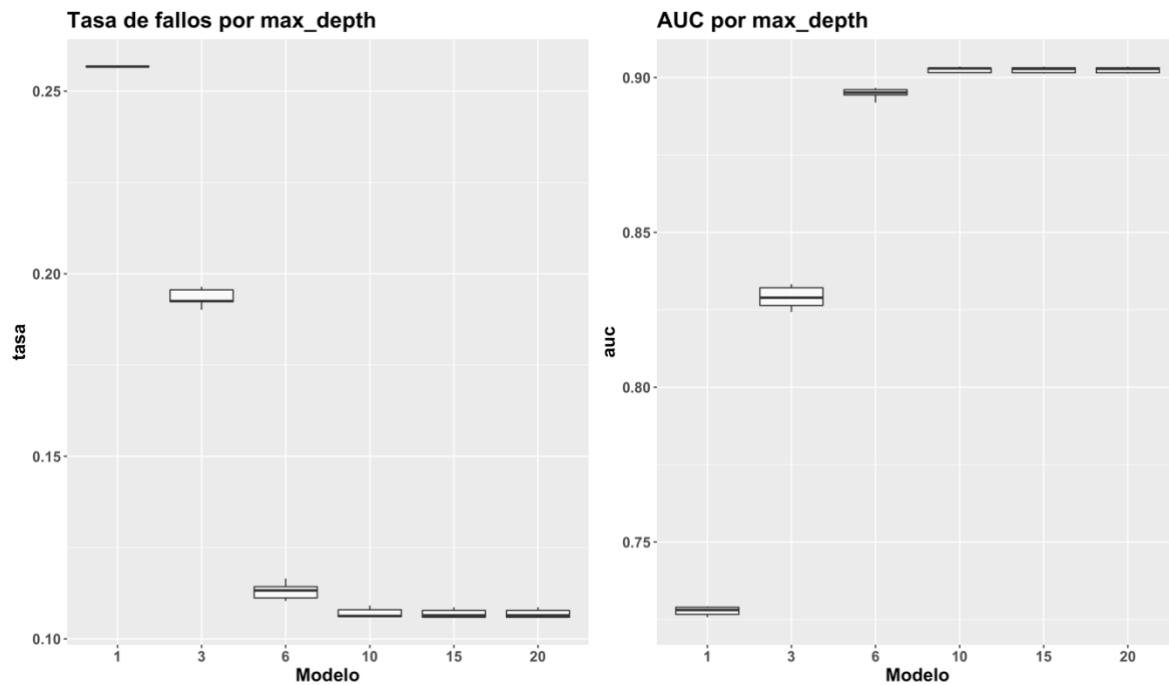


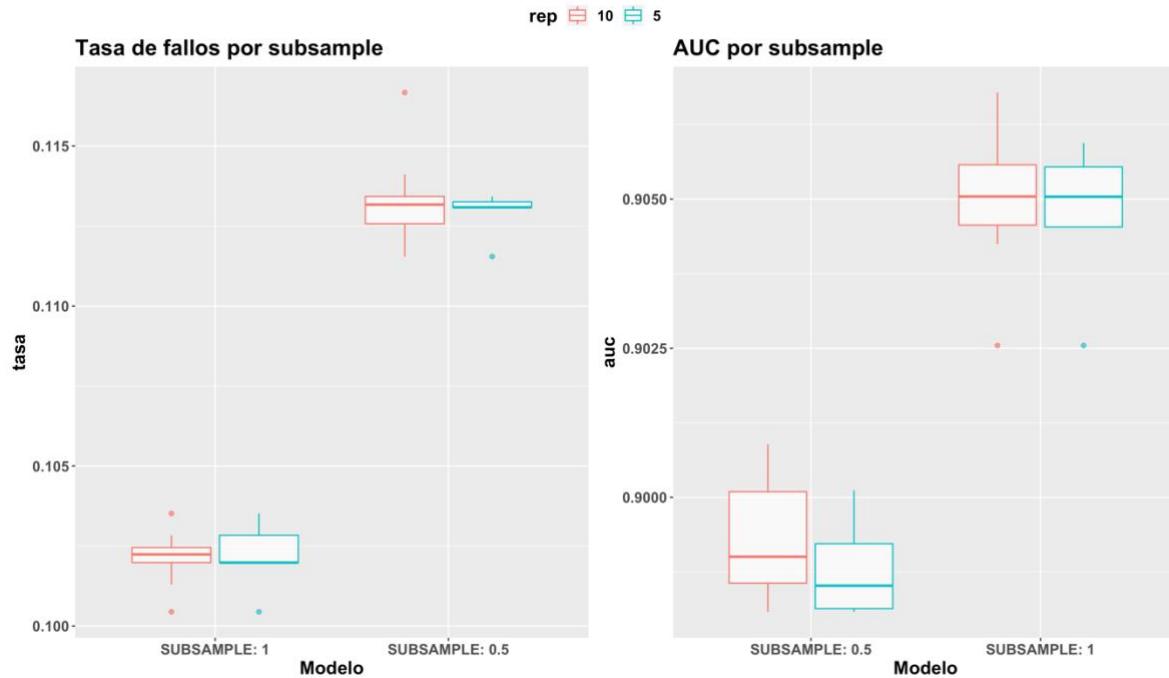
Figure 60. Tasa de fallos y AUC en función de max\_depth

Analizando los resultados, una profundidad máxima de 6 parece ser una buena opción, ya que a partir de dicho valor tanto el sesgo como el AUC se estabilizan, sin necesidad de crear árboles de mayor complejidad.

#### 11.4 Tuneo *subsample*

Como último parámetro a tunear, probamos con el parámetro *subsample* con el que sortear el número de observaciones, especialmente para comprobar cómo afecta a la varianza, más que al sesgo. Concretamente, y del mismo modo que lo realizado con *gradient boosting*, probaremos sorteando el 50 % de las observaciones (aunque no sea el valor por defecto):

**Nota:** para observar mejor el efecto del sorteo de observaciones, probamos tanto con 5 como con 10 repeticiones.



*Figure 61. Tasa de fallos y AUC en función de subsample*

En términos de varianza, al ser datos tan sencillos no se observa mejora alguna. No obstante, **no hay demasiada diferencia entre sortear el 100 % de las observaciones y tan solo la mitad**. Por tanto, mantenemos el parámetro *subsample* a 1 (valor por defecto en los paquetes *XGboost*).

#### **RESUMEN XGboost:**

1. *nrounds* = 100
2. *eta* = 0.1
3. *min\_child\_weight* = 20
4. *colsample\_bytree* = 1,
5. *subsample* = 1
6. *max\_depth* = 6

#### **11.5 Comparación final**

Del mismo modo, se mantiene en la "línea" de lo obtenido de otros modelos no lineales como *random forest*, *bagging* o *gradient boosting*: baja tasa de fallos y elevado AUC:

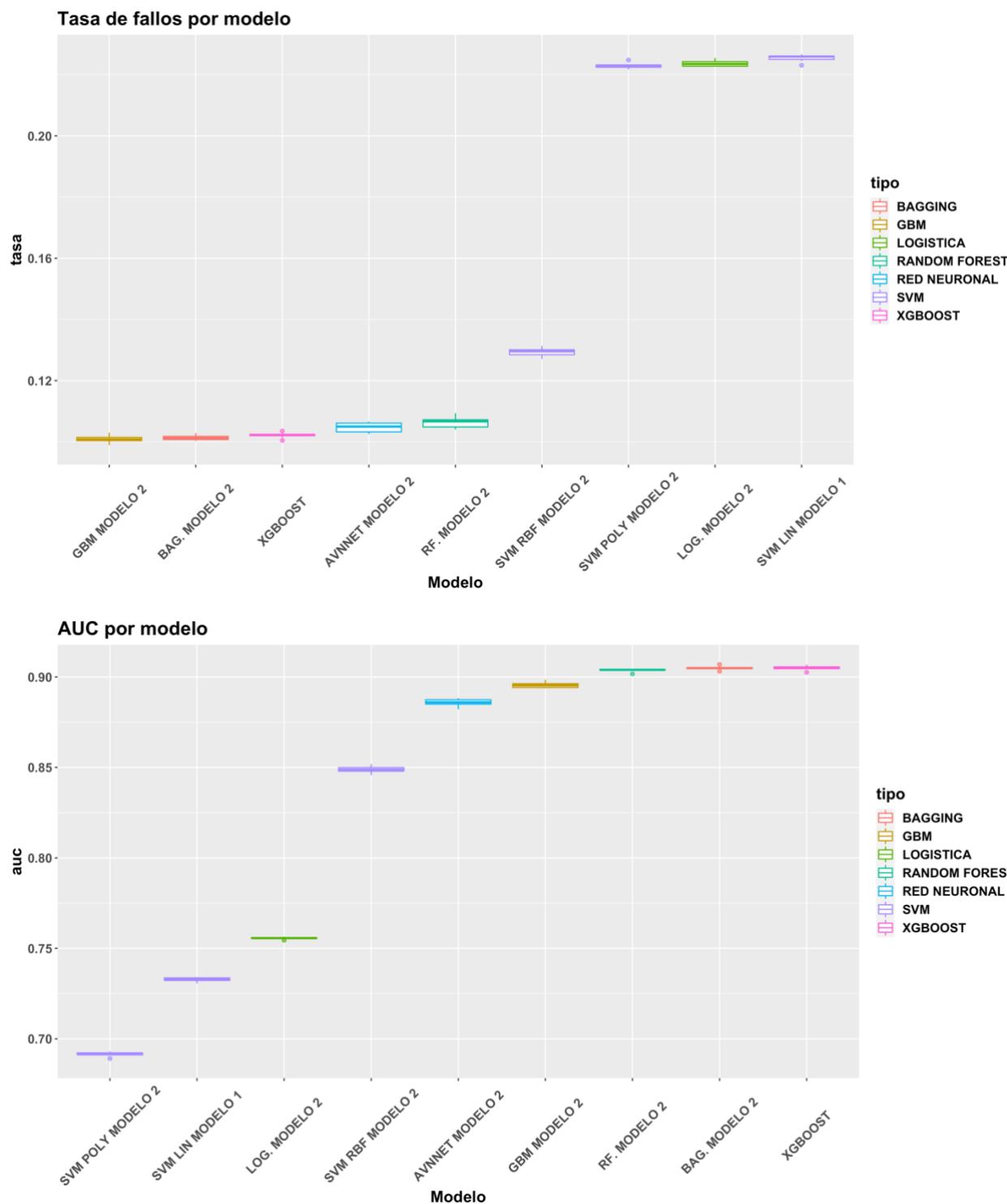


Figure 62. Comparación logística-avnnet-bagging-random forest-gbm-svm-Xgboost

## 12. Ensamblado

### 12.1 Correlación entre los modelos

Inicialmente, de forma previa al ensamblado **analicemos la correlación entre cada uno de los modelos**, entrenados mediante validación cruzada repetida del mismo modo que lo hecho hasta el momento (5 grupos y 10 repeticiones):

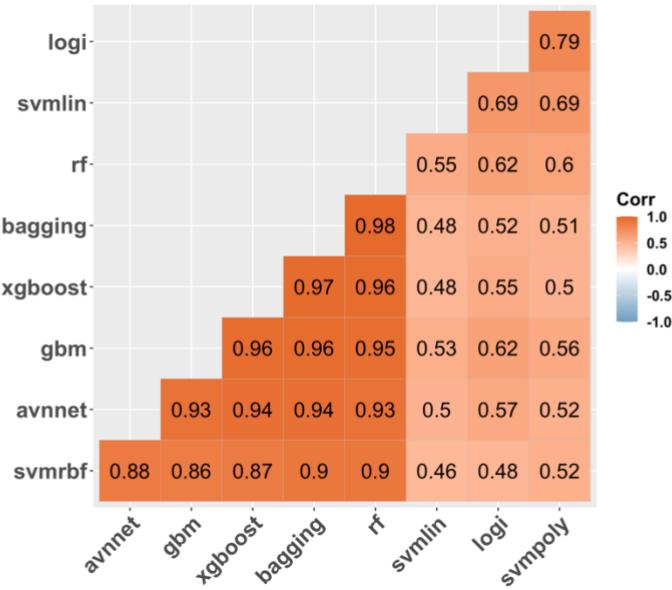


Figure 63. Correlación entre modelos

Podemos comprobar cómo las diferentes familias presentan una alta correlación entre sí, diferenciando entre dos grandes grupos:

1. **Modelos no lineales basados en árbol, como *bagging* o *random forest*, junto con los modelos basados en *boosting* como *gbm* y *xgboost*, así como la red y el *svm* con *kernel radial***, donde en ninguno de los casos la correlación disminuye de 0.8.
2. **El modelo logístico y las *svm* con *kernel* lineal y polinomial.** En cualquiera de los casos, la correlación no disminuye de 0.6.

Como era de esperar, **se encuentra generalmente una alta correlación entre modelos no lineales y lineales por separado**. Por tanto, ¿Cómo influye de cara al ensamblado? Por lo general, y desde un punto de vista teórico, existen dos factores que influyen en el error de un modelo ensamblado: **el sesgo y la correlación**: uniendo clasificadores con un sesgo lo suficientemente bajo y similar además de una correlación baja, el sesgo en el ensamblado, teóricamente, disminuirá aún más (o al menos no aumentará demasiado):

$$E_{add}^{ave}(\bar{\beta}) = \frac{s}{2} \left( \sigma_b^2 \left( \frac{1+\delta(N-1)}{N} \right) + \frac{\beta^2}{z^2} \right)$$

Donde  $\bar{\beta}$  representa el sesgo con ensamblado y  $\delta$  la correlación entre clasificadores. Por tanto, con un sesgo promedio y grado de correlación bajo, menor error en el ensamblado (teóricamente).

## 12.2 Ensamblado con dos clasificadores

Por tanto, el objetivo será unir clasificadores que presenten un sesgo lo suficientemente bajo y similar, con poca correlación entre ellos; y en caso de que el promedio del sesgo no varíe demasiado, que la correlación entre los modelos sea la más baja posible. De hecho, en base al gráfico anterior, existen varias alternativas, uniendo principalmente modelos lineales y no lineales entre sí:

- *svm* lineal, polinomial o logística con *Random forest*, *bagging*, *gradient boosting*, *Xgboost*, *avnnet* e incluso *svm* con *kernel* radial, **modelos cuya correlación entre si es moderada, en torno a 0.5-0.6, aproximadamente.**

Sin embargo, volvamos un momento a la tasa de fallos:

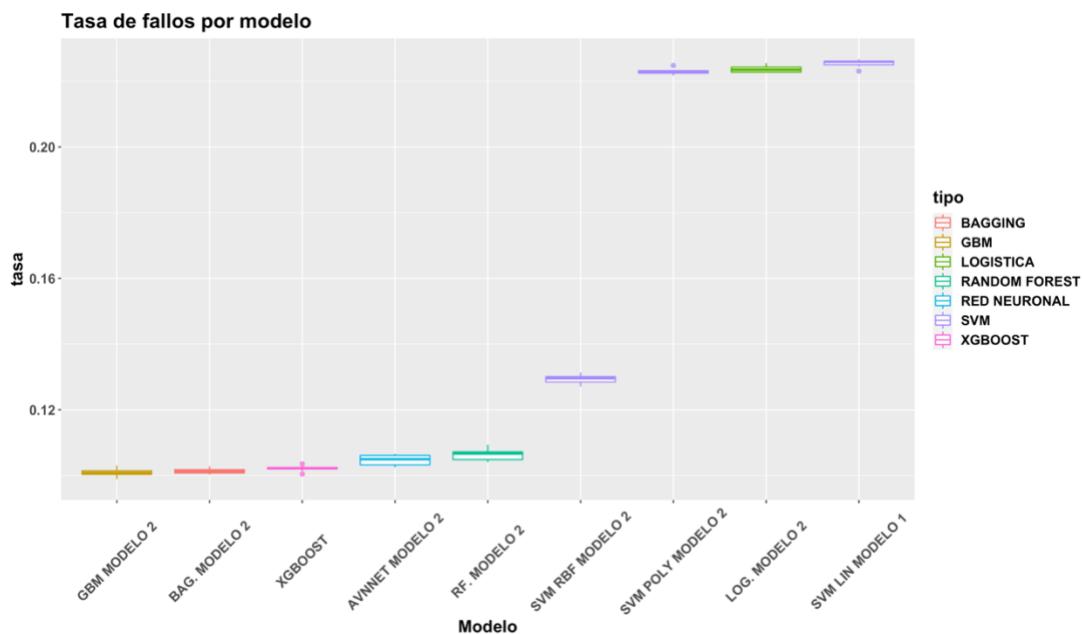


Figure 64. Tasa de fallos modelos candidatos

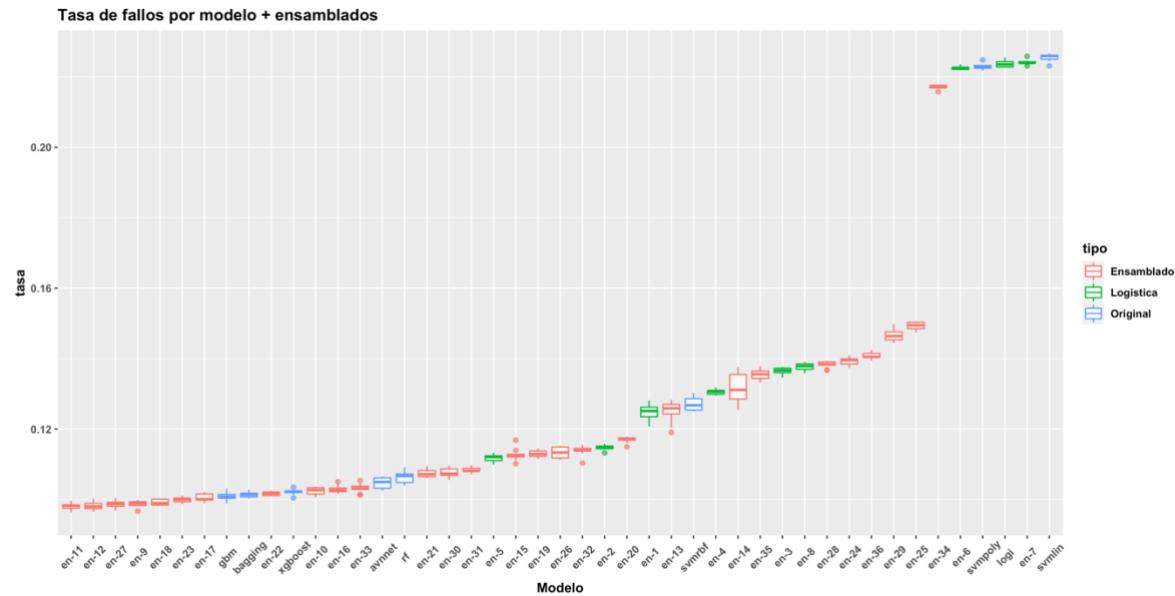
Llama la atención un aspecto fundamental: **el contraste del sesgo entre modelos con poca correlación**. A modo de ejemplo, mientras que la tasa de fallos de un modelo de árbol o *gradient boosting* se sitúa entre 0.11-0.10, aproximadamente, los modelos lineales como logística o *svm* lineal logran alcanzar un valor de 0.22. Esto último supone un inconveniente a la hora de construir modelos de ensamblado: **con un alto contraste entre sesgos, es posible que el modelo de ensamblado no logre mejorar los resultados de modelos originales como Random Forest, bagging, gbm o Xgboost.**

Sin embargo, que el sesgo mejore o no es algo que **no se puede conocer a priori**, por lo que es necesario realizar pruebas empíricas. Como consecuencia, realizamos una primera aproximación, comenzando con un ensamblado entre dos modelos, realizando todas las posibles combinaciones entre sí (mediante el promedio de resultados):

```
unipredi[, paste0("en-", i)] <- (unipredi[, modelo] + unipredi[, modelo_aux]) / 2
```

Además, dado que la regresión logística no obtuvo buenos resultados, remarcamos con **color verde** cualquier ensamblado en el que intervenga dicho modelo, observando con ello si mejora o no:

### Tasa de fallos:



### AUC:

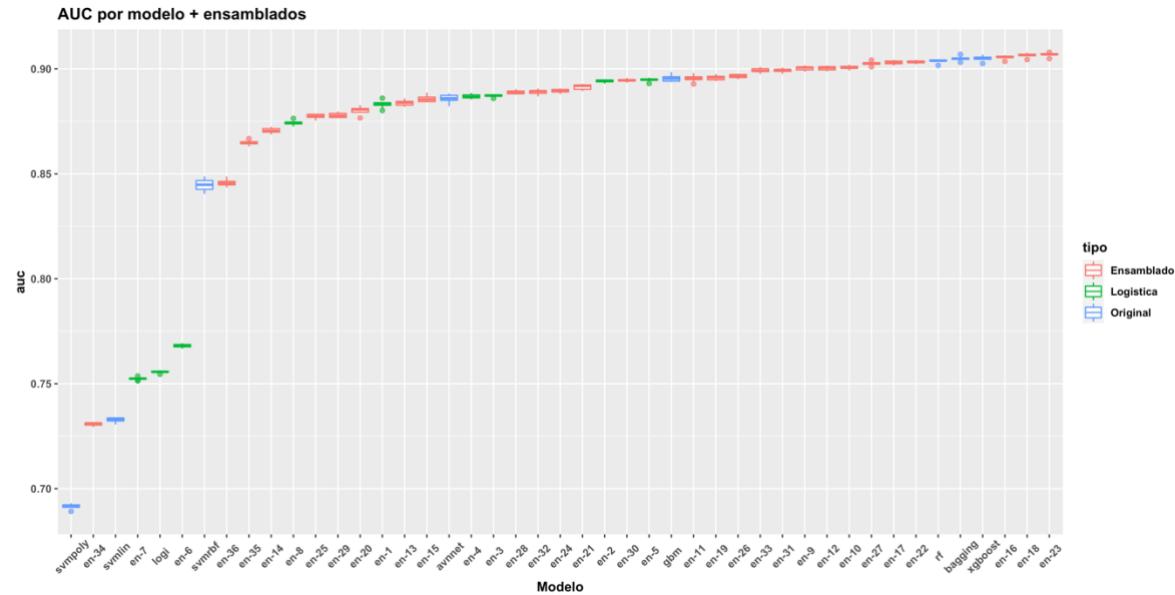


Figure 65. Tasa de fallos y AUC modelos originales + ensamblado

En primer lugar, cabe destacar que en conjunto con modelos no lineales, la **regresión logística mejora tanto en tasa de fallos como en AUC (mostramos los ensamblados de peor a mejor AUC)**:

Nombre	en-8	en-1	en-4	en-3	en-2	en-5
<i>logística +</i>	<i>svm poly</i>	<i>avnnet</i>	<i>gbm</i>	<i>random forest</i>	<i>bagging</i>	<i>XGboost</i>

Combinando con modelos no lineales, **la tasa de fallos se reduce por debajo de 0.20 y el AUC aumenta hasta 0.85**. Sin embargo, pese a su mejoría continua siendo preferible modelos no lineales como *random forest*, *bagging* o *gbm*, dado que al juntarlos con un modelo con peor sesgo, provoca que el resultado mejore (en relación con el modelo logístico original) aunque no lo suficiente como para considerarlo una potencial alternativa.

Por otro lado, debemos destacar los mejores modelos de ensamblado, tanto en tasa de fallos como en AUC:

- **Con menor tasa de fallos:**

1. *en-11: avnnet + gbm*
2. *en-12: avnnet + Xgboost*
3. *en-27: gbm + Xgboost*

- **Con mayor AUC:**

1. *en-23: Random Forest + Xgboost*
2. *en-18: bagging + Xgboost*
3. *en-16: bagging + Random Forest*

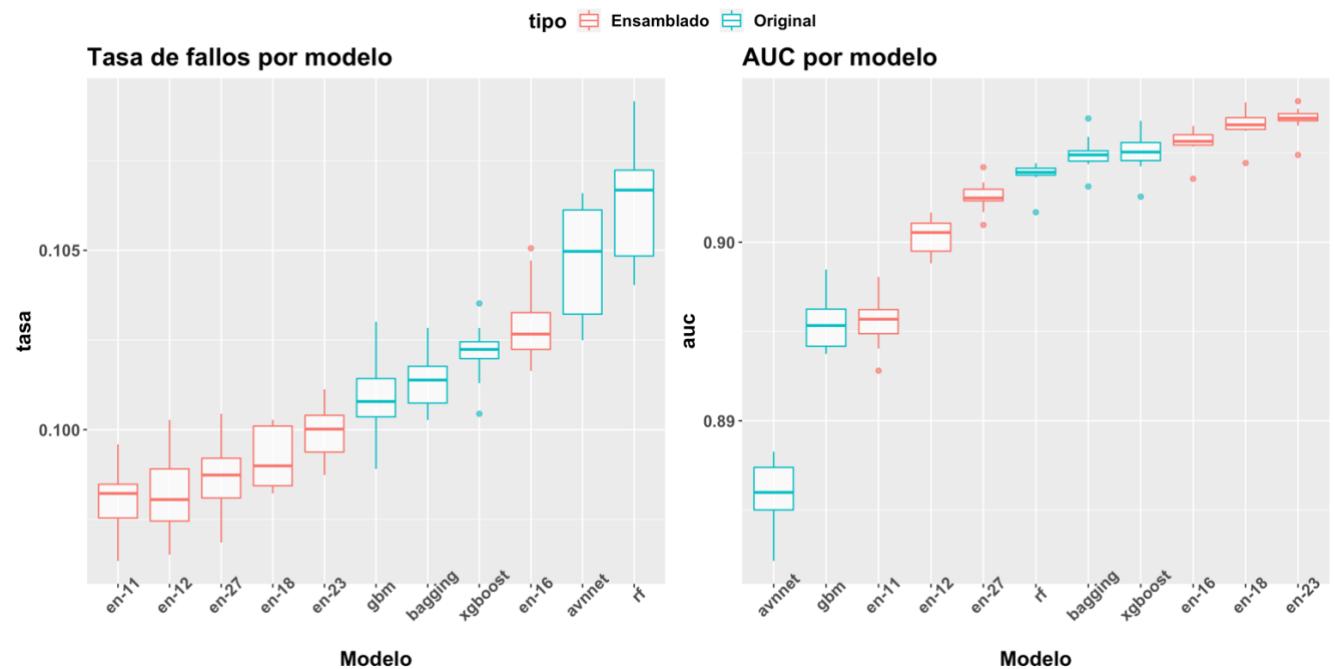
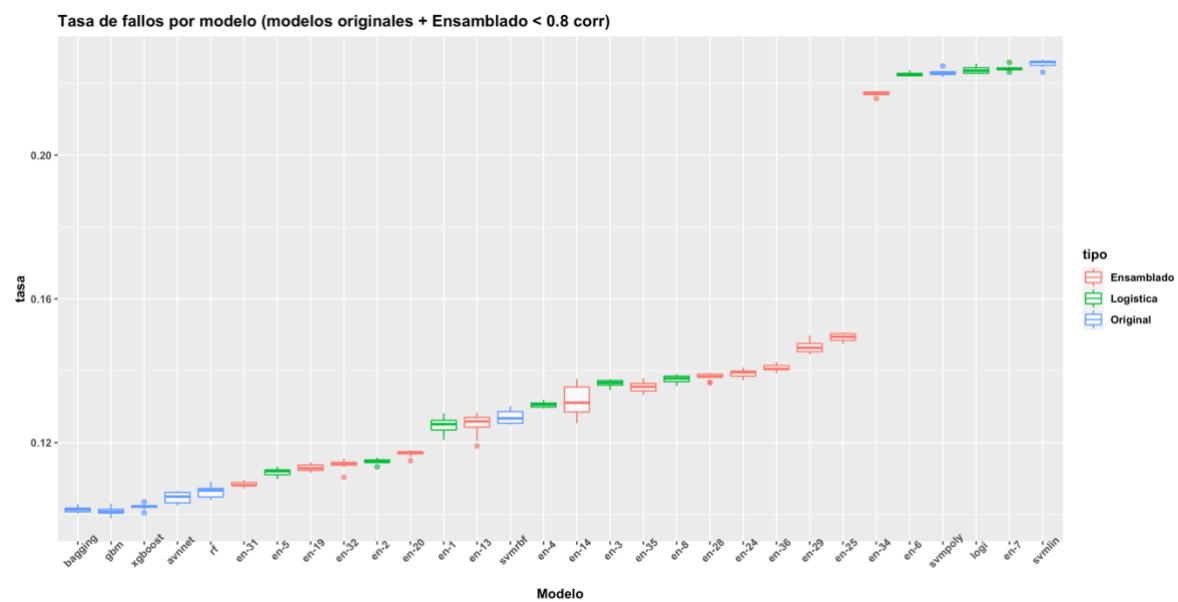


Figure 66. Comparación tasa fallos y AUC modelos originales + ensamblado (II)

Llama la atención como, tanto en tasa de fallos como en AUC, **los mejores ensamblados corresponden con modelos con sesgo similar y alta correlación entre si**. La cuestión es ¿Mejoran el error con respecto a los modelos originales? Desde un punto de vista numérico, empírico, si consigue reducir el error con respecto a los modelos originales. Sin embargo, no estamos teniendo en cuenta la escala de los ejes. Por ejemplo, uniendo avnet y gbm (**en-11**) solo conseguimos mejorar la tasa de fallos en **0.005**. Por otro lado, uniendo *Random Forest* con *Xgboost* tan solo consigue aumentar el AUC en apenas unas milésimas.

Por tanto, está claro que unir modelos con sesgo similar pero alta correlación reduce muy ligeramente el error, una diferencia tan pequeña que puede deberse a muchos otros motivos, tales como la estructura del remuestreo o incluso el valor de la semilla inicial. Por otro lado, ¿Qué sucede con el resto de los ensamblados cuyos modelos presentan poca correlación? **Si filtramos del gráfico anterior, descartando ensamblados con modelos con alta correlación (> 0.8)**:

### Tasa de fallos:



## AUC:

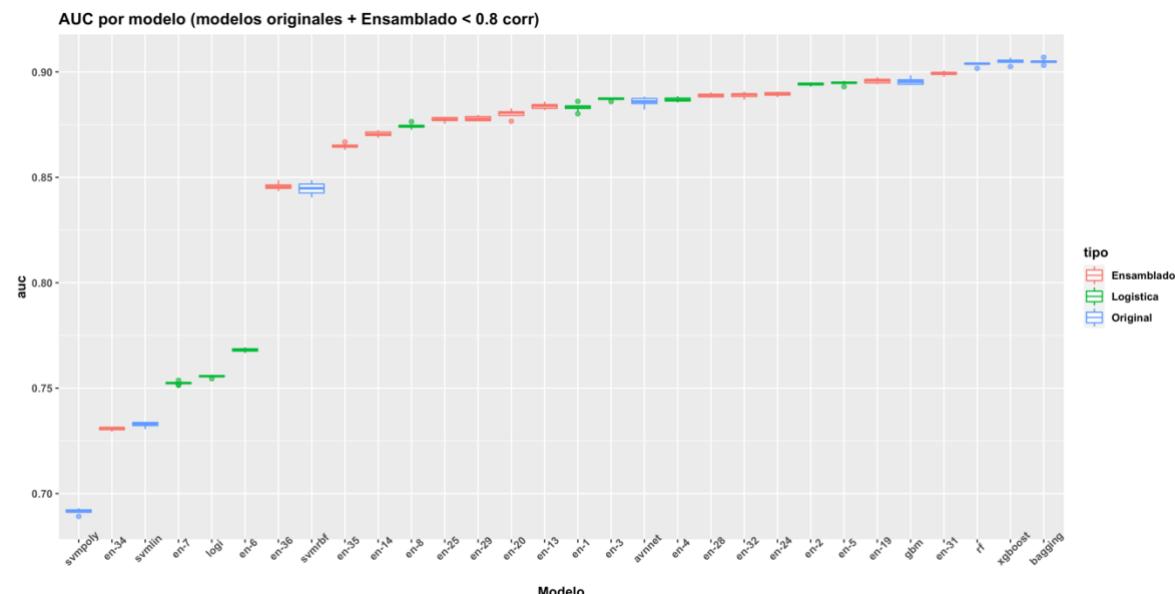


Figure 67. Tasa de fallos y AUC modelos originales + ensamblado (correlación < 0.8)

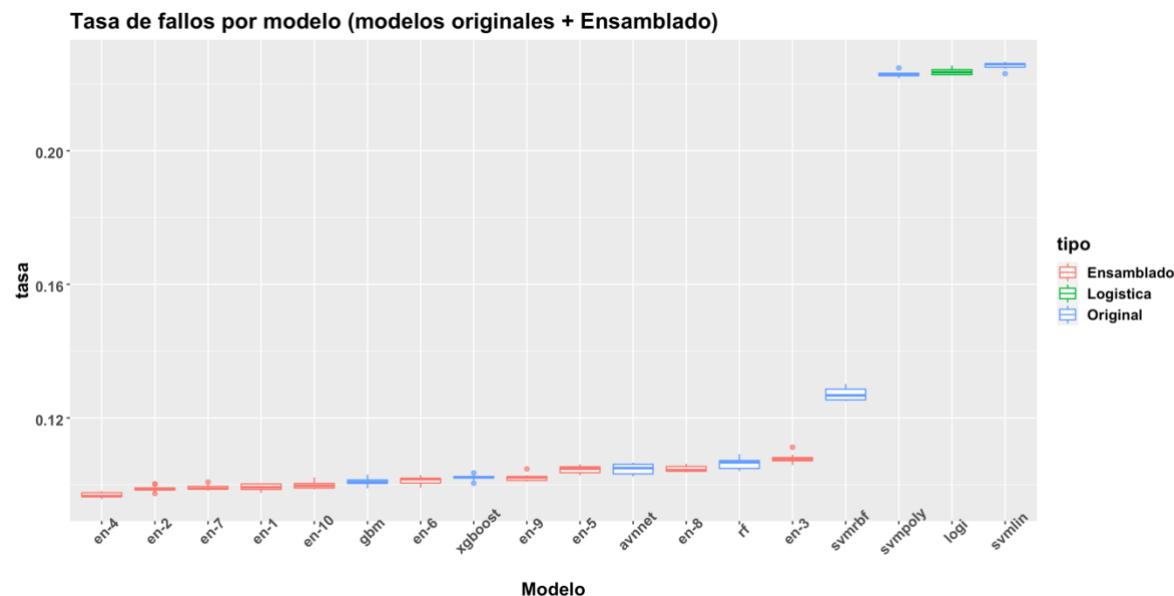
Queda demostrado lo comentado anteriormente: dado que los clasificadores con baja correlación presentan tasas de error diferentes (variación de 0.10-0.11 a 0.20-0.22), tanto en tasa de fallos como en AUC no consiguen mejorar lo suficiente como para considerarlos una alternativa frente a los modelos originales.

Por tanto, podemos concluir que **uniendo dos clasificadores no se consigue mejorar significativamente los resultados del modelo**, en especial cuando los modelos con las tasas de error más bajas son, precisamente, aquellos con una alta correlación entre sí, lo que se traduce en una mejoría poco o nada apreciable por parte de los ensamblados.

### 12.3 Ensamblado con tres clasificadores

No obstante, **podríamos probar con los mejores clasificadores en un ensamblado de tres modelos**, concretamente aquellos con alta correlación, es decir, *random forest*, *svm rbf*, *avnnet*, *gbm* y *xgboost* (*bagging* lo descartamos ya que con *random forest* se obtienen prácticamente los mismos resultados):

## Tasa de fallos:



## AUC:

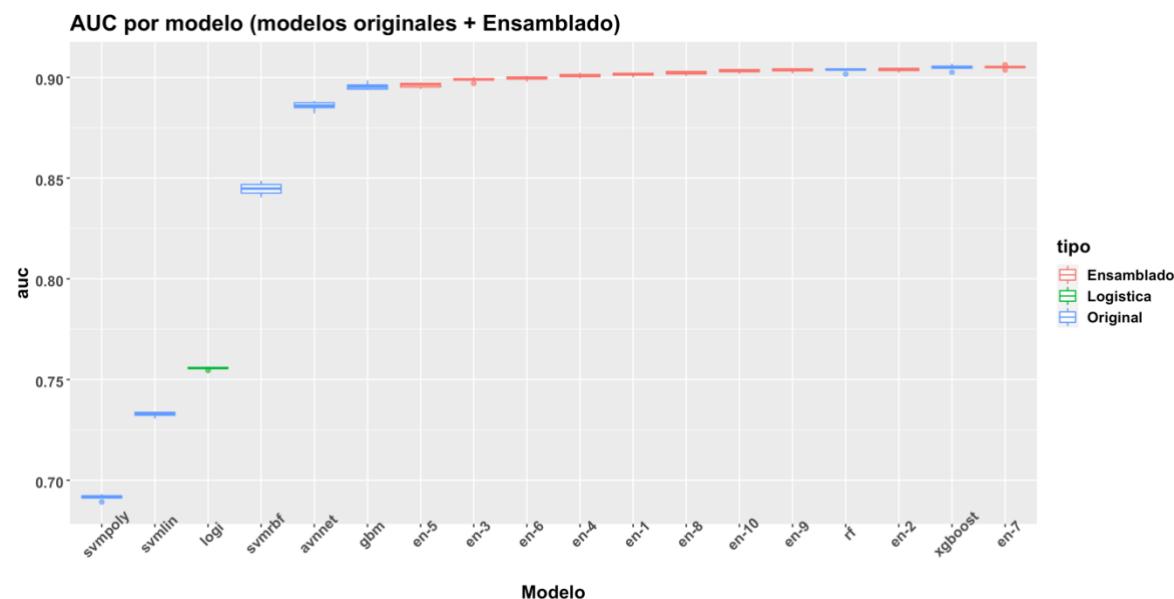


Figure 68. Tasa de fallos y AUC modelos originales + ensamblado modelos alta correlación

Pese a añadir tres clasificadores, la ganancia continúa siendo insignificante.

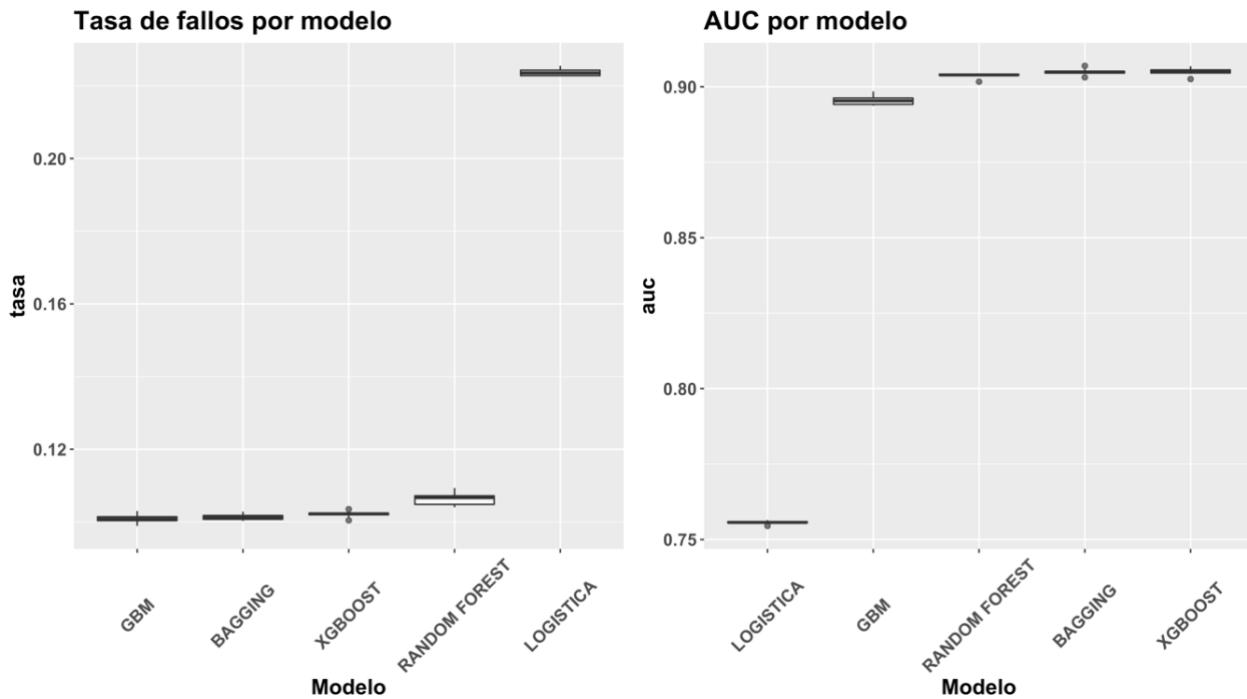
## 12.4 Conclusiones ensamblado y comparación mejores algoritmos vs logística

A la vista de los resultados obtenidos, ¿Merece la pena aplicar ensamblado? La respuesta es no, principalmente por dos motivos:

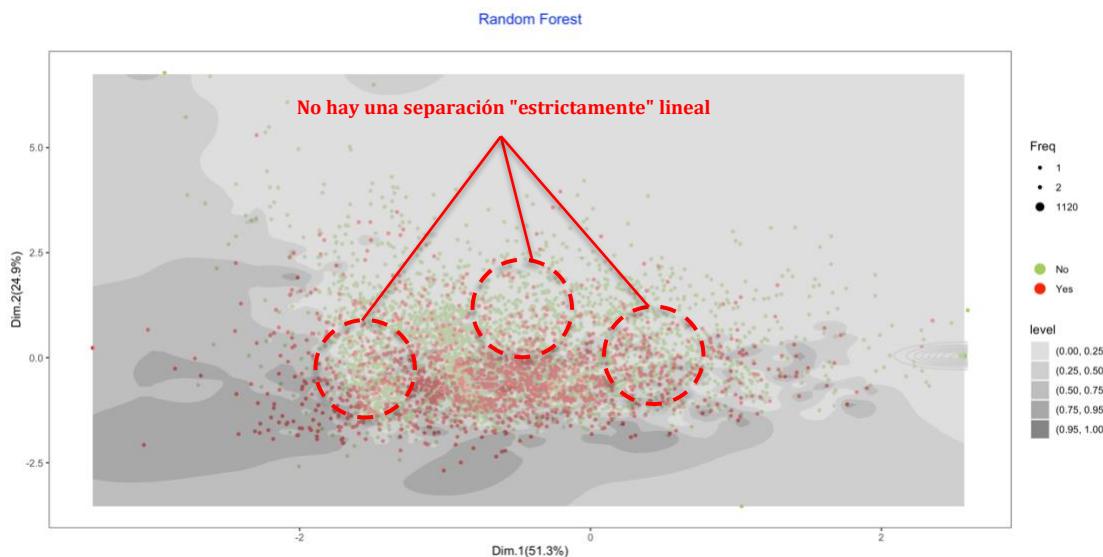
1. Los modelos originales que obtienen mejores resultados corresponden con aquellos con alta correlación, esto es, modelos de árbol, red y boosting.

## 2. Incluso ensamblando dichos modelos, la mejoría es poco o nada relevante.

Está claro que **los modelos no lineales con alta correlación son los algoritmos dominantes para este conjunto de datos**, por lo que no merece la pena el ensamblado. En conclusión, **de todos los modelos elaborados a lo largo de la práctica**, y sin necesidad de recurrir al ensamblado, los modelos **no lineales** han demostrado obtener mejores resultados en comparación con modelos tradicionales como la logística. Por tanto, **tomamos como modelos finalistas a bagging, Random Forest, gbm y Xgboost**, tal y como podemos observar en una última comparación final entre un modelo logístico y modelos basados en árbol y *boosting*:



De forma adicional, podemos observar dicha **separación no lineal** de los datos:



*Figure 69. Distribución no lineal de los datos (Random Forest)*

## 13. Comparación con *h2o*

Una vez elaborados todos los modelos, es momento de preguntarnos **¿Se cumplen las expectativas obtenidas con *h2o*?** Es decir, con *autoML* obtuvimos inicialmente (como mejor modelo) *gradient boosting* con los siguientes valores AUC:

```
##                   modelo      auc
## 1 GBM_5_AutoML_set1_variables 0.9211350
## 2 GBM_5_AutoML_set2_variables 0.9096419
```

Si los comparamos con el AUC de los mejores modelos obtenidos:

```
##                   modelo auc_medio
## 1      bagging set 1   0.9144
## 2      bagging set 2   0.9049
## 3 Random Forest set 1   0.9144
## 4 Random Forest set 2   0.9038
## 5       gbm set 1    0.9043
## 6       gbm set 2    0.8954
```

En relación con *gradient boosting*, las diferencias son pequeñas entre ambos paquetes (de 0.92 a 0.90 con el primer *set* y de 0.90 a 0.89 en el segundo), lo cual puede ser debido a la estructura del remuestreo o la optimización de los algoritmos en los diferentes paquetes. Por tanto, lo obtenido por *h2o* se aproxima a los resultados de *caret*.

Además, si nos fijamos en los parámetros de ambos paquetes, (tanto *h2o* como *caret*), resultan ser bastante similares, en relación con el número de árboles (82-72 frente a 100), además de que con *caret* sorteamos tan solo la mitad de las observaciones mientras que *h2o* sortea el 80 % :

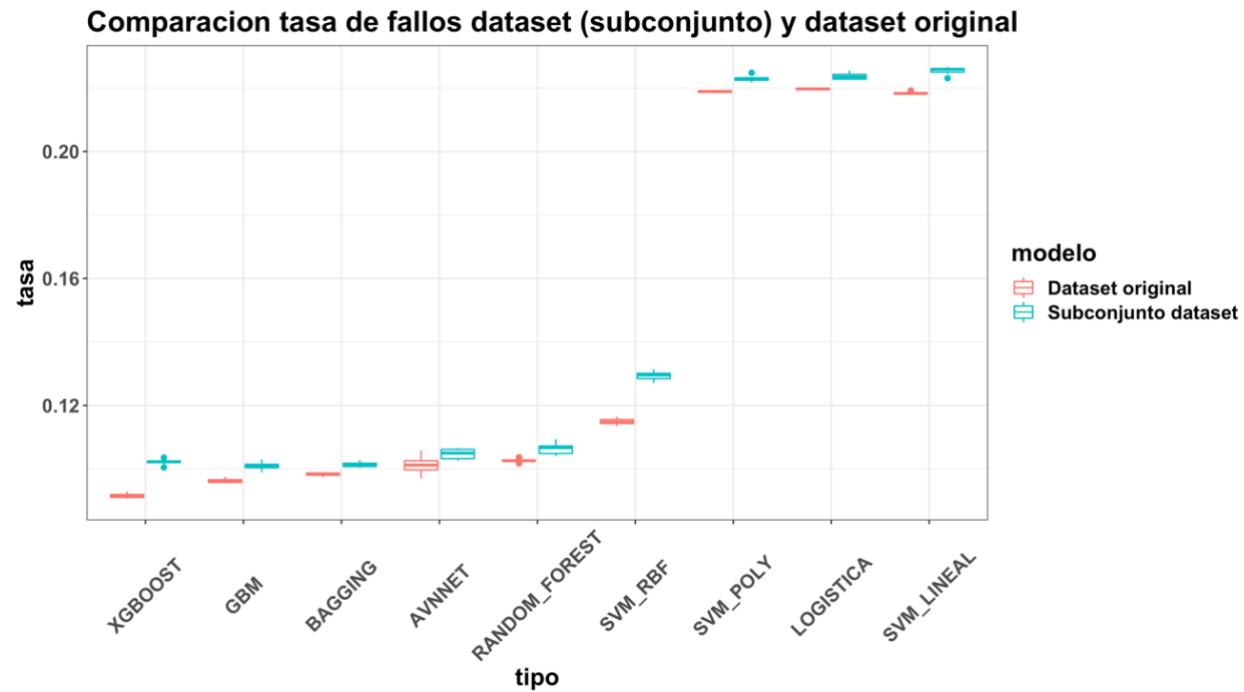
```
##                   modelo ntrees sample_rate
## 1 h2o Modelo 1     82        0.8
## 2 h2o Modelo 2     72        0.8

##                   modelo ntrees bag.fraction
## 1 caret Modelo 1    100        0.5
## 2 caret Modelo 2    100        0.5
```

## 14. Probando con el *dataset* completo

A continuación, y utilizando únicamente el segundo *set* de variables (salvo con el modelo *svm* lineal, con el que empleamos el primer *set*), **probamos los mismos algoritmos "tuneados" con el conjunto de datos original, comprobando si el orden entre algoritmos se conserva**:

## Tasa de fallos:



## AUC:

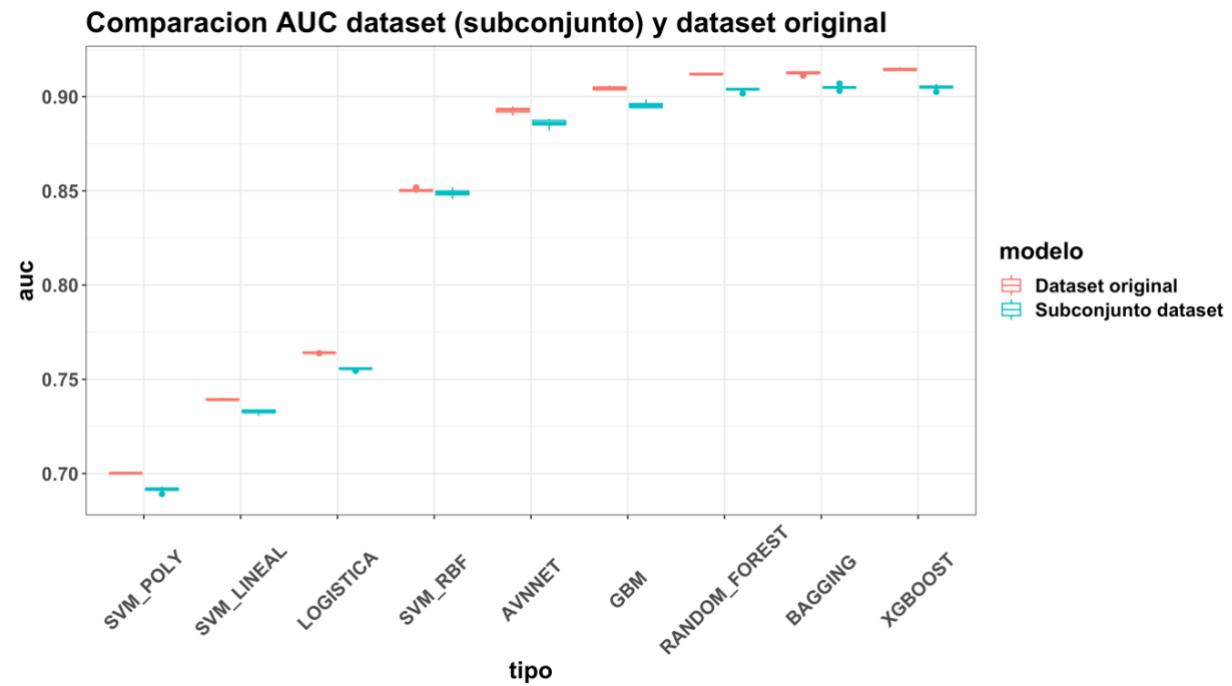


Figure 70. Comparación tasa de fallos y AUC dataset original vs subconjunto

De forma general, el orden de los modelos, tanto con el **dataset original** como con el **subconjunto** utilizado durante la práctica, se conserva, un indicativo de que no ha sido estrictamente necesario emplear todas las observaciones para generalizar un modelo.

## 15. Aumento del número de grupos y repeticiones

A continuación, con los modelos finalistas (*bagging*, *gbm*, *random forest* y *XGboost*), realizamos una última comparación bajo un número alto y diferente de semillas y un número mayor de grupos de CV. De este modo, lo que se pretende buscar es que el orden de las "cajas", así como la varianza de los modelos se mantenga lo más similar posible al orden obtenido.

Dado que durante la práctica se ha trabajado con 5 grupos y un máximo de 10 repeticiones, aumentamos a 10 grupos y 20 repeticiones, además de aumentar la semilla inicial a 123456:

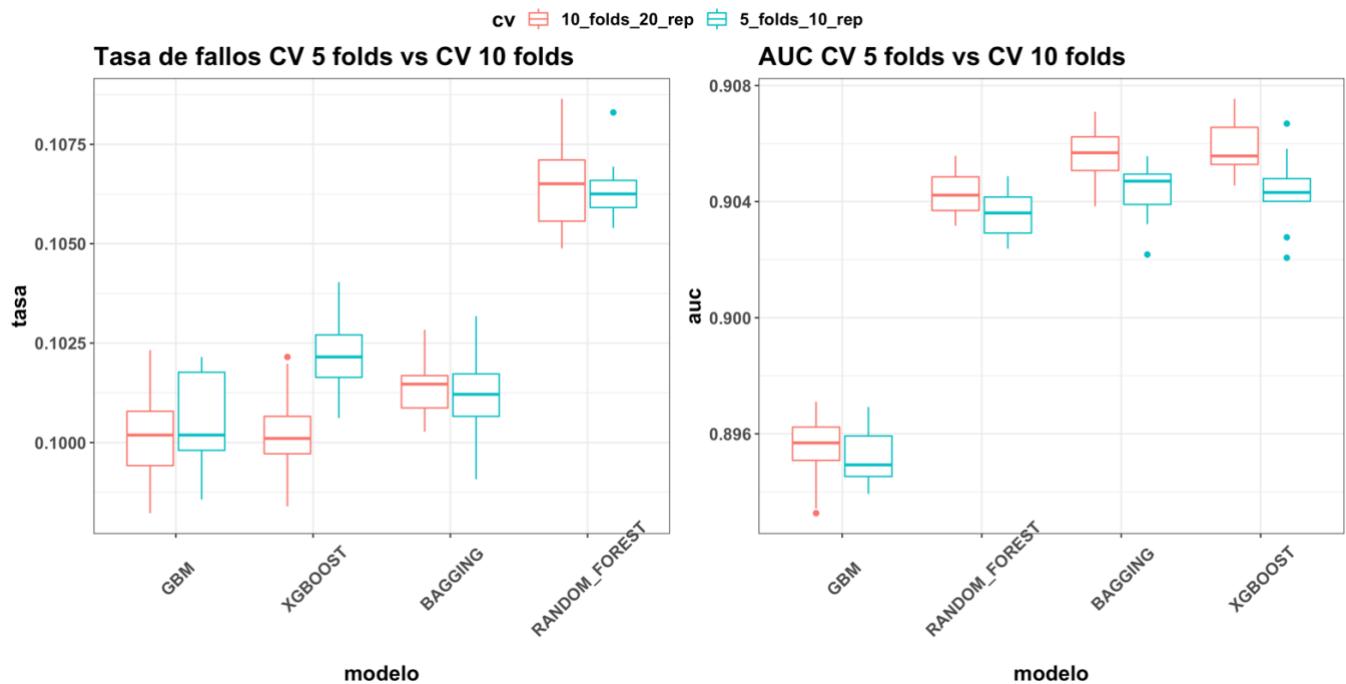


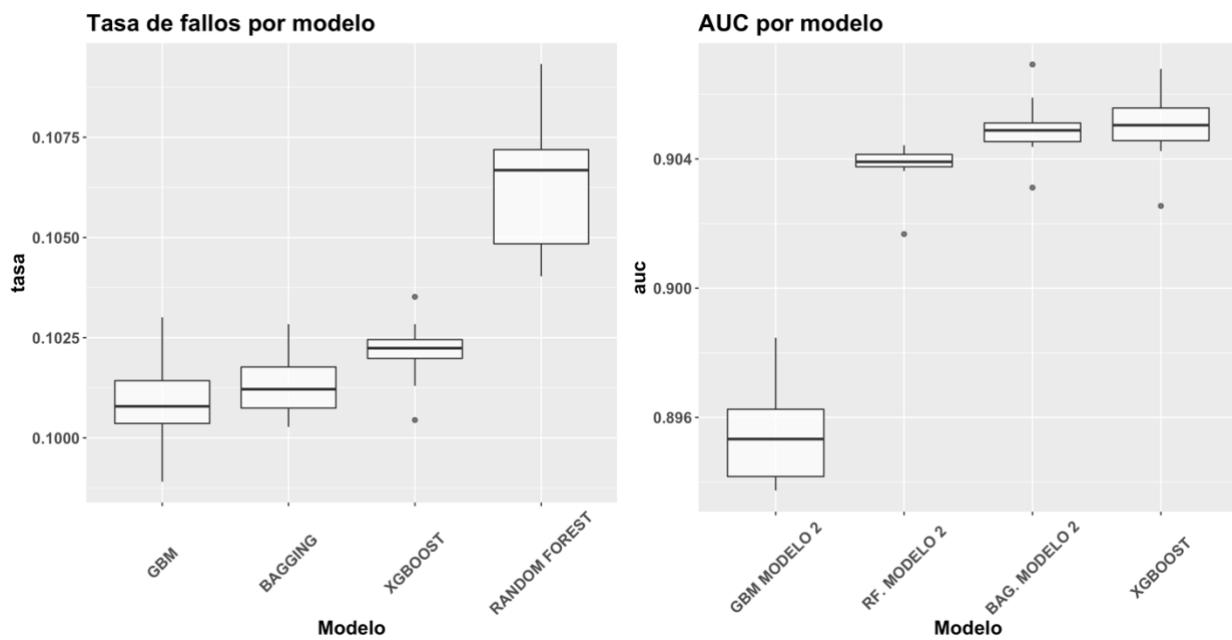
Figure 71. Comparación AUC y tasa de fallos con 5 y 10 grupos

A la vista de los resultados, tanto el orden como la varianza se mantiene similar, de forma aproximada, al aumentar el número de grupos, aunque en algunas "cajas" observemos una mayor varianza con 10 grupos, como es el caso de *Random Forest* o *Bagging* en AUC. No obstante, y dada la escala de los ejes, la diferencia de la varianza es muy pequeña, por lo que consideramos que se mantienen prácticamente idénticos.

## 16. Variación del punto de corte

Como última sección, con los mejores modelos obtenidos analizamos los valores de especificidad y sensibilidad con el punto de corte actual (0.5), además de comprobar como varían dichos algoritmos bajo diferentes puntos.

En primer lugar, de todos los mejores modelos obtenidos, ¿Por cual o cuales nos decantamos? Echemos un vistazo al AUC y tasa de fallos de los cuatro más importantes: *gradient boosting*, *bagging*, *Random Forest* y *Xgboost*.



*Figure 72. Comparación modelos finales*

De todos los modelos presentes, y dejándonos guiar por los gráficos, diríamos que el mejor en cuanto a AUC y tasa de fallos serían modelos basados en *boosting* como *gbm* y *XGboost*, pues no solo presentan bajas tasas de error, sino que además (y en especial con *XGboost*, un alto valor en el área bajo la curva).

De nuevo, la escala del gráfico puede llevar a engaño, **y es que la diferencia entre los modelos basados en árboles y *boosting* es ínfima**. A modo de ejemplo, entre la tasa de fallos de *gbm* y un modelo *Random Forest* es de tan solo 0.007. En relación con la varianza sucede lo mismo: aunque *Random Forest* aparente tener la mayor variabilidad en cuanto a tasa de fallos, lo cierto es que la amplitud es de apenas unas milésimas.

Por tanto, y a la vista de los resultados obtenidos, **podemos decantarnos por dos posibles soluciones, cuya diferencia no es muy relevante: o bien un modelo basado en árbol o basado en *boosting***. No obstante, analicemos las medidas obtenidas por cada modelo, esto es, **especificidad, sensibilidad, valor predictivo positivo y negativo**:

##	modelo	especificidad	sensibilidad	valor_pred_pos	valor_pred_neg
## 1	gbm	<b>0.9851</b>	<b>0.6487</b>	<b>0.9375</b>	<b>0.8903</b>
## 2	Random Forest	<b>0.9763</b>	<b>0.6487</b>	<b>0.9044</b>	<b>0.8894</b>
## 3	bagging	0.9841	0.6474	0.9338	0.8899
## 4	Xgboost	0.9805	0.6593	0.9210	0.8928

En cualquiera de los modelos, **las medidas obtenidas son muy similares**, modelos cuyo principal "inconveniente" es la mayor tasa de falsos negativos que presentan, lo cual se traduce en un porcentaje de sensibilidad bajo. Es decir, mientras que los modelos son capaces de clasificar correctamente a un paciente sin complicación (alto porcentaje de especificidad), el número de falsos negativos **dificulta clasificar a un paciente con complicaciones**, es decir, como verdadero positivo. Si a lo anterior se le añade el desbalanceo de clases sobre la variable objetivo, el porcentaje de sensibilidad disminuye (también el valor predictivo positivo, aunque en menor porcentaje dado el mayor número de pacientes "sin complicaciones").

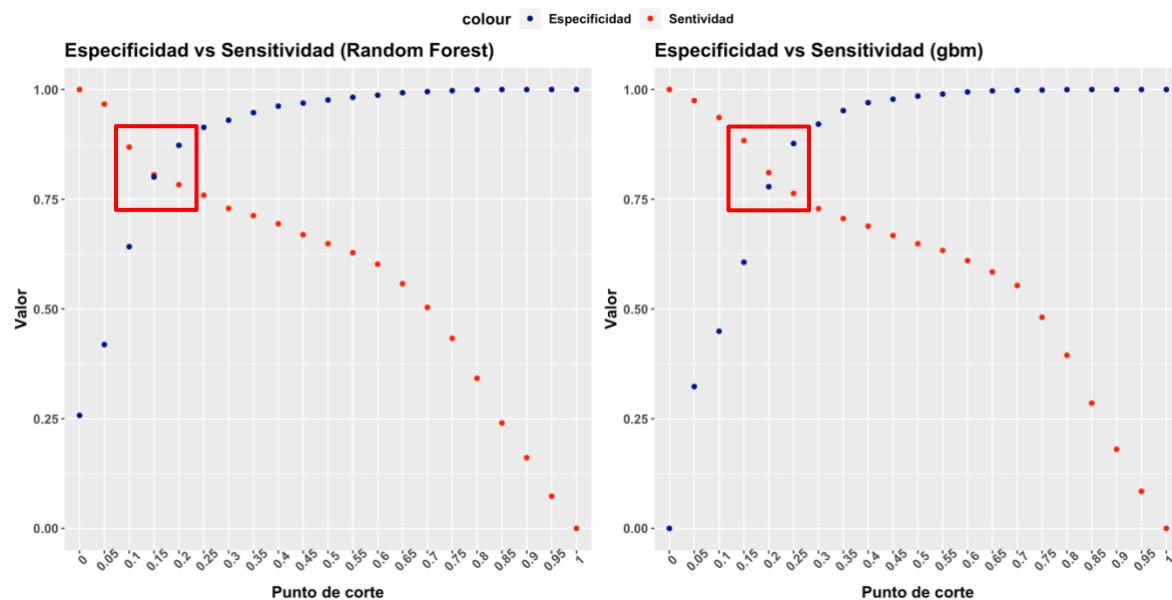
Por tanto, ¿Qué modelo basado en árbol y *boosting* escogemos? En este caso, y dado que la diferencia no es especialmente relevante, podemos decantarnos por:

1. **Modelos de árbol:** una buena opción sería escoger *Random Forest*, dado que a diferencia del modelo *bagging* **no sortea todas las variables en cada nivel del árbol**, sino tan solo 2. Además, la ganancia de error que presenta con respecto a la tasa de fallos es ínfima.
2. **Modelos *boosting*:** dado que no existe mucha diferencia entre *gbm* y *XGboost* (tan solo apenas unas centésimas de diferencia en tasa de fallos y AUC), **nos decantamos por *gradient boosting***, en lugar de su variante más reciente.

## 16.1 Medidas básicas (punto de corte = 0.5)

A continuación, procedemos a variar el punto de corte para los modelos *Random Forest* y *gbm*, con saltos de 0.05 y validación cruzada de 5 grupos:

```
puntos_corte <- seq(0, 1, 0.05)
```



*Figure 73. Comparación puntos de corte random forest - gbm*

Analizando los puntos de corte, sobre ambos modelos **cabe destacar el uso de un punto de corte bajo, en torno a 0.15, 0.2**, aproximadamente; para equilibrar las tasas de sensibilidad y especificidad, donde ambas medidas se sitúan en torno al 75-80 %. Con el punto de corte original, 0.5, los valores originales de sensibilidad no son especialmente bajos (0.66), por lo que está claro que es mejor utilizar uno de los dos modelos que una simple predicción aleatoria:

##	modelo	punto_corte	VP	FP	VN	FN	sensibilidad	especificidad
## 1	Random Forest	0.50	975	103	4248	528	0.6487026	0.9763273
## 2	Random Forest	0.20	1177	553	3798	326	0.7831005	0.8729028
## 3	Random Forest	0.15	1211	867	3484	292	0.8057219	0.8007355
## 1	gbm	0.50	975	65	4286	528	0.6487026	0.9850609
## 2	gbm	0.20	1218	963	3388	285	0.8103792	0.7786716
## 3	gbm	0.15	1328	1712	2639	175	0.8835662	0.6065272

En general, **el número de verdaderos positivos aumenta a medida que reducimos los puntos de corte**, además de reducirse el número de falsos negativos, con su consiguiente aumento en el porcentaje de sensibilidad. No obstante, disminuir demasiado el punto de corte puede afectar a la especificidad. A modo de ejemplo, tan solo reduciendo de 0.2 a 0.15, en el caso de *Random Forest* pasamos 553 a 867 FP y de 3798 a 3484 VN; e incluso en *gradient boosting* el cambio es más pronunciado: **aumentando a más de 700 FP**, con el consiguiente empeoramiento en el porcentaje de especificidad.

La cuestión es ¿Interesa priorizar la detección de falsos positivos o por el contrario conseguir un equilibrio sin perder de vista la especificidad? **Si lo que se desea es un equilibrio sensibilidad-especificidad (tanto rf como gbm) un buen punto de corte sería 0.2, pues no solo mejoraría el número de verdaderos positivos y falsos negativos, sino que mantendría incluso una alta tasa de verdaderos negativos: entre un 78-81 % de sensibilidad y un 87-77 % de especificidad, respectivamente.** Por el contrario, si nos fijamos en el punto que maximiza el índice de Youden:

$$J = \text{especificidad} + \text{sensibilidad} - 1$$

```
# Random Forest
dataframe_puntos_corte_rf_multiple_seeds[which(indice_youden_rf == max(indice_youden_rf)), ]
  punto_corte    FP   FN    VP    VN      AUC      tasa sensibilidad especificidad
1       0.25  360 362 1141 3991 0.905239 0.1233345     0.7591484     0.9172604

# gbm
dataframe_puntos_corte_gbm_multiple_seeds[which(indice_youden_gbm == max(indice_youden_gbm)), ]
  punto_corte    FP   FN    VP    VN      tasa sensibilidad especificidad
1       0.4   130 468 1035 4221 0.1021524    0.6886228     0.9701218
```

**Si el objetivo es mejorar la sensibilidad sin perjudicar la tasa de especificidad, un punto de corte de 0.25 en *Random Forest* es también una buena alternativa, frente a un modelo *gbm* (sensibilidad en torno al 68 % frente al 75 % de *rf*).**

## 16.2 Prueba con diferentes semillas

Por último, y con el simple objetivo de comprobar la estabilidad de los resultados, **lanzamos ambos modelos con una validación cruzada de 5 grupos, variando la semilla aleatoriamente**:

```
-- Probamos con diferentes semillas aleatorias (10)
semillas <- sample(1:10000, size = 10)
```

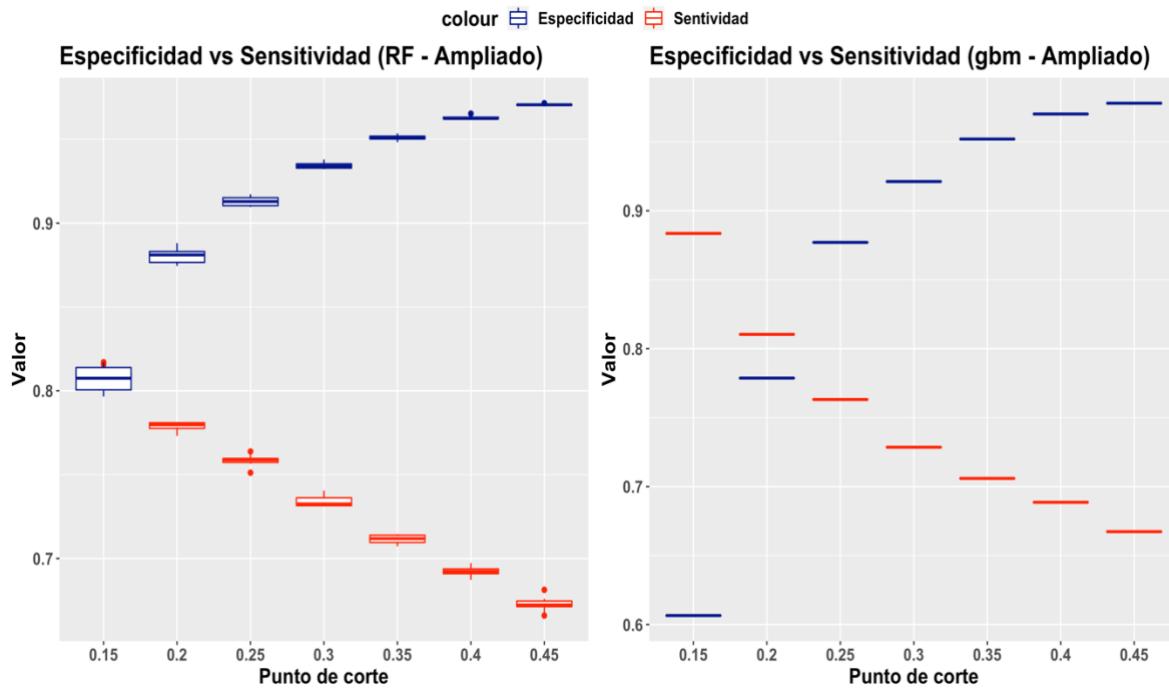


Figure 74. Puntos de corte gbm - variando semillas entre 0.15 y 0.45

Incluso variando el número de semillas, **las medidas obtenidas son prácticamente estables en ambos modelos.**

## 17. Conclusiones

A lo largo de la práctica, hemos analizado (con los mejores parámetros y *sets* de variables) la calidad de los diferentes algoritmos de clasificación, utilizando dos posibles *sets* de variables candidatos, de los cuales (dada la escasa o nula diferencia entre ambos), **nos hemos acabado decantando por el más simple (4 variables), siguiendo el principio de parsimonia: ante dos teorías en igualdad de condiciones, la más sencilla suele ser la más probable.**

No obstante, en el último apartado (con la selección del punto de corte), no nos hemos decantado aún por un modelo en particular. Está claro que los modelos no lineales resultan ser claramente los mejores en cuanto a AUC y sesgo se refiere, teniendo como finalistas dos modelos como *Random Forest* y *gbm*. Sin embargo ¿Cuál debemos elegir?

### 17.1 Mejor modelo (punto de vista computacional)

Desde un punto de vista computacional, ambos son buenos modelos:

1. **Sortean observaciones** (un 17 % en *Random Forest* y un 50 % en *gbm*).
2. **En el caso de *Random Forest***, sortea dos variables en cada nivel.
3. **En el caso de *gbm*, tan solo emplea 100 iteraciones.**

Cada uno de ellos presenta sus características, pero en relación con los resultados (tasa de fallos y AUC), ambos presentan buenos resultados. Incluso comparando ambos modelos en otros lenguajes como es el caso de *Python* (aplicando los mismos hiperparámetros con la librería *sklearn*):

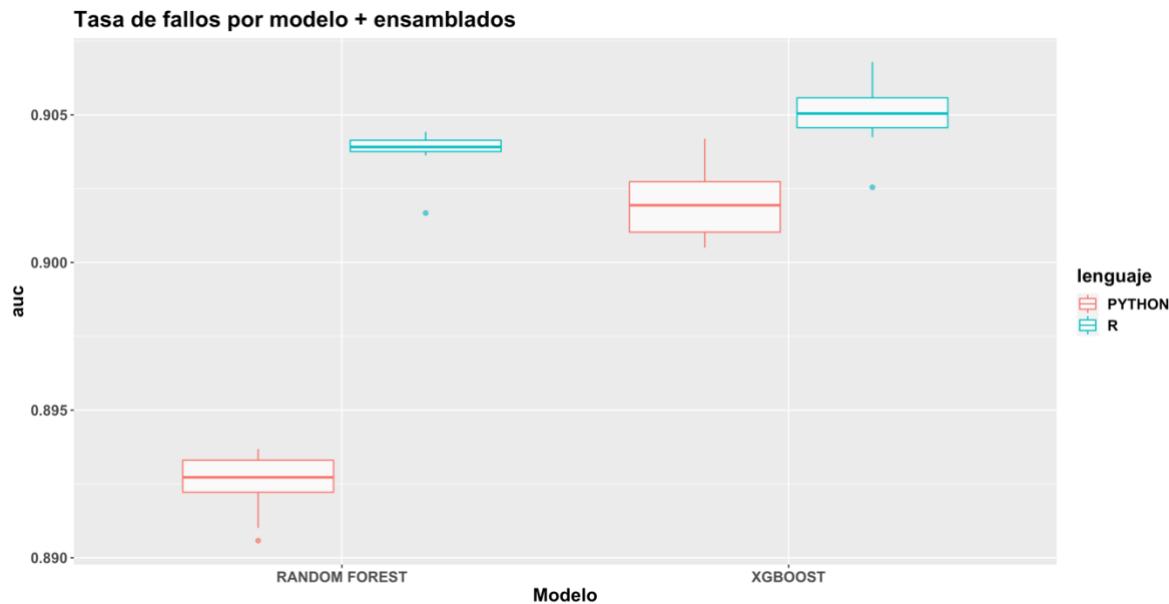


Figure 75. Comparación Random Forest y gbm - R vs Python

Aunque no exactos (dada la estructura de remuestreo y el uso de diferentes librerías y lenguajes) se aproximan bastante a los obtenidos con R, tanto en valor como en varianza, donde podemos comprobar nuevamente que apenas oscila en unas milésimas de diferencia.

## 17.2 Mejor modelo (punto de corte)

Por otro lado, si analizamos los puntos de corte en ambos modelos, observamos que el equilibrio sensibilidad-especificidad es muy similar en ambos casos (punto de corte = 0.2):

```
##           modelo punto_corte sensibilidad especificidad
## 1 Random Forest      0.20     0.7831005    0.8729028
## 2          gbm        0.20     0.8103791    0.7786716
```

Por tanto, si el objetivo inicial del proyecto es un mayor aumento de la sensibilidad (pese a la reducción de la tasa de especificidad), cualquiera de los modelos, con un punto de corte 0.2, sería una buena opción por la que decantarse (aunque parece existir una mayor tasa de sensibilidad con *gradient boosting*). Sin embargo, maximizando el índice de *Youden*, podemos decantarnos por *Random Forest*: aumenta la sensibilidad, sin disminuir la especificidad del 90 %.

```
# Random Forest

##           modelo punto_corte sensibilidad especificidad
## 1 Random Forest      0.25     0.7591484    0.9172604
## 2          gbm        0.4      0.6886228    0.9701218
```

En conclusión, y como opinión final, ambos modelos son buenos candidatos por lo que no me decantaría por un modelo en específico: todo dependerá del objetivo principal de la

investigación, si priorizar una mayor detección de pacientes con posibles complicaciones o si, por el contrario, lograr un equilibrio entre ambas clases, o al menos no afectar en gran medida a la especificidad.

### 17.3 Árbol básico y tabla coeficientes logística

Por último, finalizamos la práctica elaborando un modelo de árbol sencillo (mediante *rpart*), además de obtener los coeficientes del modelo logístico, todo ello con el mejor *set* de cuatro variables:

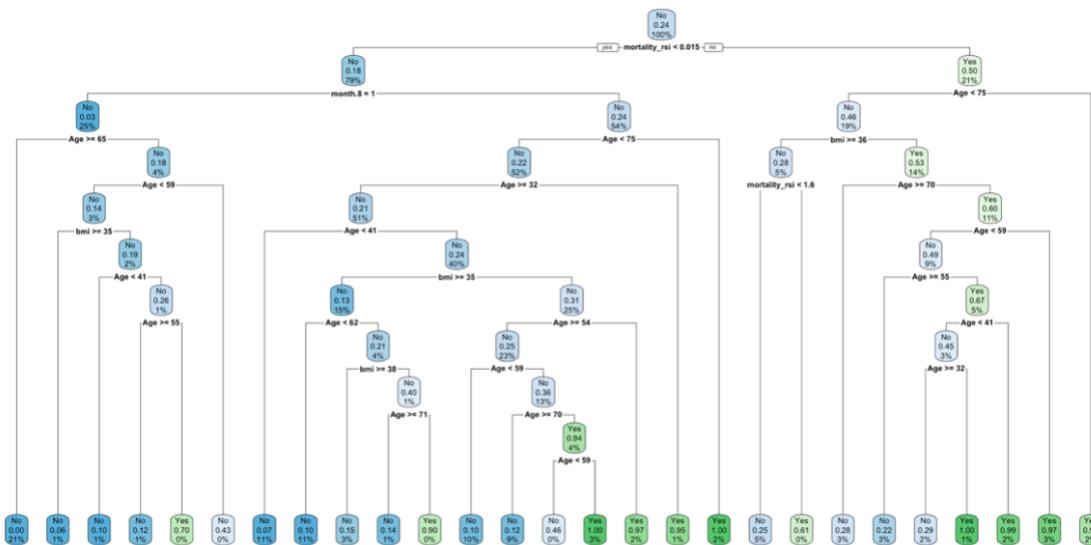


Figure 76. Árbol básico con *rpart* (datos sin estandarizar) - *minbucket* = 40 + grafico importancia

Analizando el árbol, podemos comprobar que todas las variables empleadas participan en el modelo. Sin embargo, variables como *mortality\_rsi* o en especial ***Age*** son las variables más utilizadas para establecer los diferentes puntos de corte en cada nivel de árbol, dado que (tal y como pudimos comprobar en el apartado de depuración en el caso de la edad), no existía una clara separación lineal, a partir de una cierta edad, entre pacientes con o sin complicaciones hospitalarias, realizando por ello múltiples puntos de corte sobre varios tramos de edad.

Por otro lado, destacan otras variables como *month.8*, la cual se emplea para separar, en el primer nivel, aquellos pacientes que no estuvieron ingresados durante el mes de septiembre, etiquetándolos como pacientes "sin complicaciones", algo que pudimos incluso comprobar al comienzo de la práctica (analizando el valor de información), donde se observaba que en el mes de septiembre se acumula una mayor proporción de pacientes sin complicaciones. Por el contrario, *bmi* es la variable menos empleada (de hecho, los puntos de corte, aunque los datos estén sin estandarizar, se realizan sobre el mismo punto: 35-36)

Por otro lado, en relación con el modelo logístico, resulta de gran importancia estudiar los coeficientes resultantes, y en especial su signo:

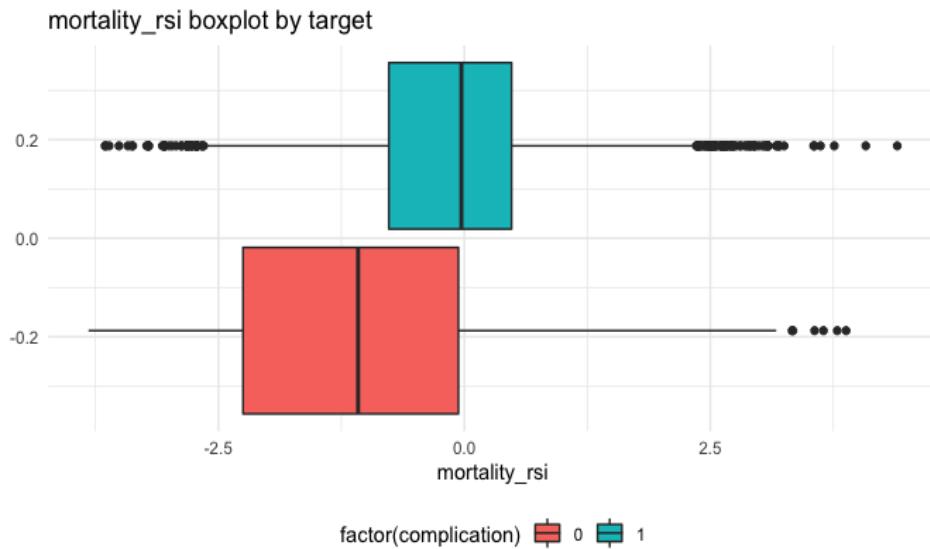
```
##      (Intercept)      Age  mortality_rsi    month.8        bmi
## -0.9672613  0.2726029   0.7819655 -1.3814499 -0.3643954
```

- En el caso de variables como *Age* o *bmi*, si analizamos el signo de cada coeficiente, obtenemos lo siguiente:

- *Age* (coeficiente positivo): 0.27, por cada unidad en la que se incrementa el parámetro de edad, la ODD de que un paciente sufra una complicación postoperatoria aumenta en  $e^{0.27} = 1.30$ .
- *bmi* (coeficiente negativo): -0.36, por cada unidad en la que se decremente el índice de masa corporal, la ODD de que un paciente sufra una complicación hospitalaria es de  $e^{-0.36} = 0.69$ .

¿Esto último significa que aquellos pacientes con mayor edad y/o menor índice de masa corporal son necesariamente más propensos a padecer una complicación hospitalaria? No necesariamente, dado que pudimos comprobar en la *Depuración* que existen pacientes, ya sean de edad avanzada o con un *bmi* bajo, que no sufren ninguna complicación (existe un solapamiento entre ambas cajas, por lo que no se puede esclarecer a simple vista una separación lineal).

- En el caso de la variable *mortality\_rsi*, dado su coeficiente positivo (0.78), según el modelo logístico el ODD de que un paciente sufra una complicación hospitalaria es de  $e^{0.78} = 2.18$  ¿Diríamos que los pacientes con mayor índice de riesgo de mortalidad a los 30 años son más propensos a padecer complicaciones? Si analizamos los diagramas de cajas del fichero original:



*Figure 77. Distribución mortality\_rsi según la variable objetivo*

Bien es cierto que, desde un punto de vista general, hay un mayor promedio de pacientes con mayor índice de mortalidad que sufren complicaciones (media superior), aunque la separación no es perfecta (nos encontramos con pacientes con un índice de riesgo muy bajo que, aunque se traten de *outliers*, casos atípicos, sufren complicaciones tras una intervención).

- En relación con el mes de septiembre (*month.8*), su coeficiente negativo refleja lo analizado en el apartado de depuración: el ODD de que un paciente sufra una complicación hospitalaria si el mes en el que fue operado corresponde con septiembre es de  $e^{-1.38} = 0.25$ , frente a otros a meses. Por tanto, se refleja que un paciente es menos propenso a sufrir una complicación, según los datos recabados, si el mes corresponde con septiembre.