Machine Learning

Santiago Breogán Pérez Pita - santibreo@gmail.com

09 junio, 2019

Contents

1 Introducción

Para realizar el trabajo he elegido los datos de la competición titulada *Titanic: Machine Learning from Disaster* de la plataforma Kaggle. La elección se debe a que se trata de un dataset que presenta cierta complejidad (variables categóricas y numéricas, valores missing, varias posibilidades de encontrar patrones internos), además no es un dataset de dimensiones excesivas (891 observaciones y 12 variables) lo que me permetirá entrenar los modelos con cierta agilidad y el problema que plantea es una clasificacíón binaria como exige el enunciado.

El objetivo será por la tanto predecir a partir de información de los pasajeros si estos sobrevivieron o no a la catástrofe del Titanic. Además podré subir los resultados a Kaggle, para comprobar si los modelos son tan buenos como dicen sus métricas.

2 Limpieza de Datos

En primer lugar debo indicar que al no estar este módulo centrado en el tratamiento de los datos no voy a determe en exceso en la etapa de limpieza. Como ya he dicho el set contiene información de los pasajeros. Las variables originales son:

- **PassengerId**: Un identificador del pasajero (ordinal).
- **Survived**: Si sobrevivió. 0 = No, 1 = Si
- **Pclass**: Clase en la que viajó: 1 = primera, 2 = segunda, 3 = tercera
- **Name**: Nombre completo del pasajero.
- **Sex**: Sexo del pasajero. male = hombre, female = mujer
- **Age**: Edad en años.
- **SibSp**: Número de maridos/esposas o hermanos/as a bordo.
- **Parch**: Número de padres/madres o hijos/as a bordo.
- **Ticket**: Número de ticket (identificador del billete de abordo)
- **Fare**: Tarifa asociada al pasajero.
- **Cabin**: Identificador de camarote.
- **Embarked**: Puerto en el que embarcó: Q = Queenstown, C = Cherbourg, S = Southampton

2.1 Reformateo

La variable **Cabin** no esta expresada de una forma útil para trabajar, ya que viene expresada como un número o como una letra más un número, por lo que voy a separar la letra de los números, así podré considerar una variable categórica y la otra numérica. Además en este proceso me he dado cuenta de que en uno de los casos la letra es: T. Como esto no ocurre para los datos test, voy a poner este caso igual a aquellos que no llevan letra, que son los más frecuentes (los que no tienen letra se agrupan en la categoría U). Además hay algunos valores que presentan 2 o incluso 3 números de cabina, todos ellos aparecerán representados por el primero. Finalmente como hay muchos valores sin número de cabina he generado también una variable binaria en función de si se tiene número o no.

Lo mismo ocurre con **Ticket**, aparece identificado con un código alfanúmérico, así que repito la transformación. Para las letras voy a quedarme con la parte previa a la barra inclinada (/), en caso de que esta aparezca. A priori el número de opciones que aparecen para la parte no numérica es muy elevado (25), con varias categorías infrarrepresentadas, y un alto número de *missings* (al igual que para **Cabin**, aunque he generado

su propia categoría para ellos), más adelante trataré de solventar estos problemas. Como tán solo hay 4 *missings* para la parte numérica los he puesto a cero.

Name puede parecer una variable meramente identificativa, sin embargo, contiene mucha información. En primer lugar podemos extraer el título del pasajero mediante la abreviatura que viene al principio, y luego, mediante el apellido, podemos hacernos una idea de las familias que viajaban juntas (obviamente no todas las personas con el mismo apellido son familia). Así puedo fabricar una variable de frecuencias del apellido, y mirar la tasa de supervivencia del grupo. He encontrado que solo el 17.6% de los pasajeros siguieron un destino diferente al promedio de su grupo (de entre aquellos cuyo grupo tenía algún otro individuo).

Con estas operaciones hemos extraido toda la información que me parece relevante para una limpieza de datos que tampoco pretende ser exhaustiva.

2.2 Imputación de *missings*

En este caso lo primero que he hecho es mirar cuantos datos *missing* había en cada variable, descubriendo que únicamente las variables **Age**, **Embarked** y **CabNumber** (incluída por mi) presentaban alguno.

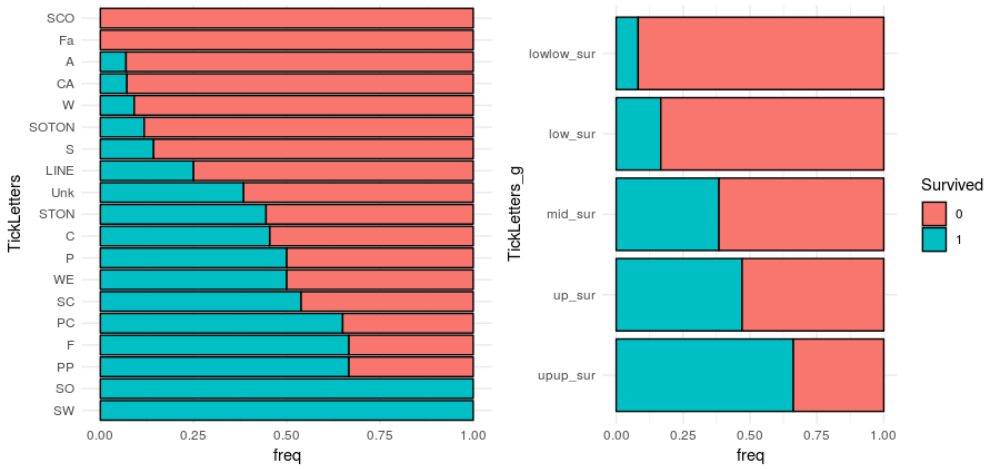
Age la he completado utilizando valores similares a los que presentaban aquellos pasajeros que pagaron una tarifa (**Fare**) similar.

Por su parte, **Embarked**, únicamente tiene vacíos dos valores, que corresponden a dos señoras mayores que viajan juntas (ya que presentan el mismo número de cabina) y que ambas sobrevivieron. Les he asignado el puerto de embarcación que más se repite entre las personas con su misma letra de camarote y sin letra en el ticket.

Para la variable **CabNumber** tengo muchas reservas ya que presenta un 77.6% de datos *missing*. Cualquier inputación resultaría en que me he inventado más de tres cuartas partes de la información contenida en la variable. Además estos valores proceden de la variable **Cabin**. Si miramos las variables **Cabin**, **CabLetter** y **CabNumber** (las dos últimas generadas por mí), hay cuatro observaciones que son *NA* en **CabNumber** y que no son cadenas de caracteres vacías en **Cabin** y son aquellas que valen: D y T. Como pese a considerar esto el número de observaciones *missing* es muy alto, me voy a decantar por usar la versión binaria **CabNumber_bin** que vale 0 si no tenemos número y uno si sí.

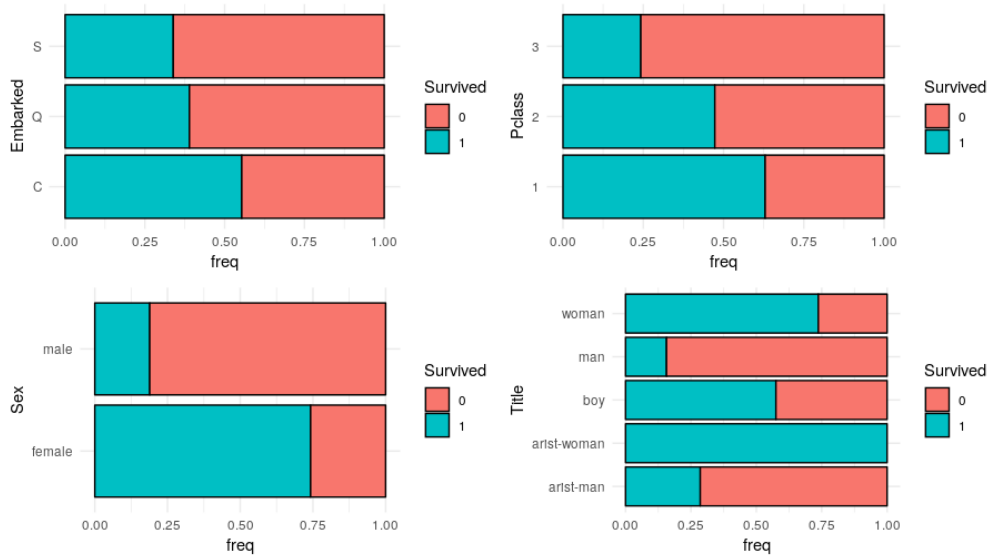
2.3 Agrupación de categorías

Únicamente la variable **TicketLetter** presenta más de 10 categorías, luego es la única que voy a agrupar. La he agrupado según el ratio de supervivientes, como muestra el siguiente gráfico:



De esta forma he reducido 17 categorías a 5, y como vemos bastante útiles a la hora de discriminar la variable **Survived**.

Podemos también mirar como se reparten los supervivientes entre el resto de factores, con la intención de tener una idea del valor clasificatorio de nuestras variables.



Vemos que el sexo es una variable discriminatoria muy fuerte, así como también lo es el título y la clase en que se viajó, únicamente el puerto de embarcación presenta menos diferencia entre el reparto, pero si hay una diferencia sensible entre las personas que embarcaron en Cherbourgh y el resto.

2.4 Eliminación de variables

Una vez completado el proceso de genaración de variables voy a eliminar de los datos aquellas que no pueden ser digeridas por ningún modelo, ya sea porque son cadenas de caracteres (como el nombre), porque adoptan excesivos valores diferentes y no son agrupables (como el apellido), porque presentan demasiados *missings* (como el número de camarote) o porque no se van a poder calcular para el set test, como el ratio de supervivientes por apellido (**SurnSurvival**). Además eliminaré el ticket y la cabina, ya que su información la he repartido en 2 variables nuevas. Hecho esto he añadido una variable aleatoria a los datos, para determinar la influencia del resto y he guardado los datos dispuesto a comenzar la selección de variables.

3 Selección de variables

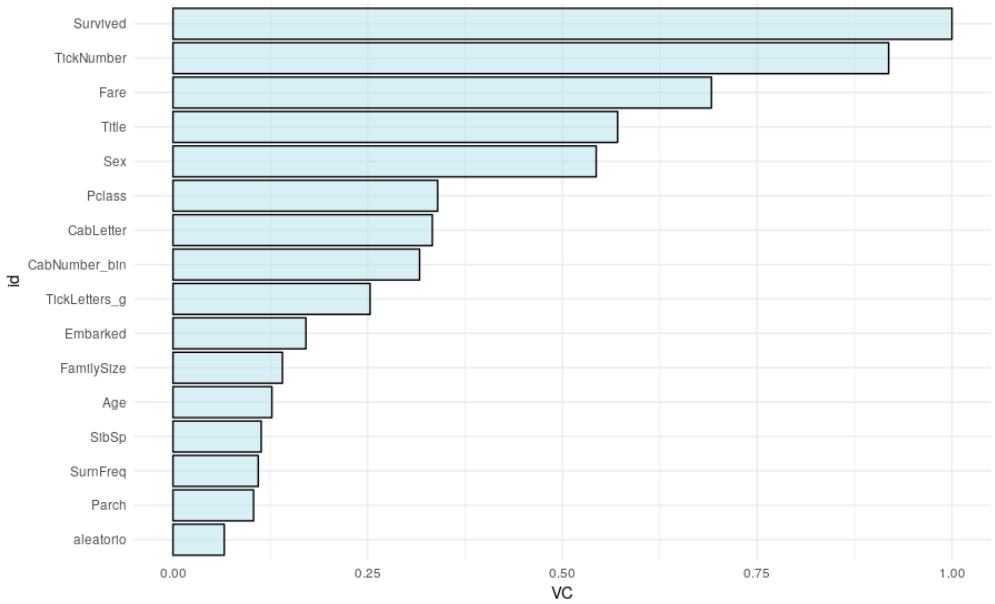
Voy a usar 3 criterios de selección:

- La **V de Cramer**: Es un estadístico que se determina mediante la siguiente expresión.

$$\Phi_C = \sqrt{\frac{\chi^2}{N \cdot (\min(k,l) - 1)}}$$

$$\text{siendo: } \chi^2 = \sum_{i=1}^{k}\sum_{j=1}^{l} \frac{\left(n_{ij} - \frac{n_i n_j}{n}\right)^2}{\frac{n_i n_j}{n}}$$

  Y nos da una idea de la "correlación" entre dos variables, solo que siendo una de ellas categórica. Podemos representar así la similitud entre la variable objetivo y el resto y ver que queda de la siguiente forma.



A la vista del gráfico se puede comprender que el conjunto de variables seleccionadas según este criterio hayan sido: **Sex**, **Fare**, **TickNumber** y **Title**.

- Los **criterios de Akaike (AIC) y Schwarz (SIC)**: Se trata de unos criterios que penalizan al modelo según este va incorporando variables. Sabemos que R^2 se puede mejorar indefinidamente incorporando variables al modelo, para solventar esta problemática se establecieron estos dos criterios de información que penalizan según el modelo va aumentando el número de parámetros que requiere. Matemáticamente se expresan:

$$AIC = n \cdot \log\left(\frac{SSE}{n}\right) + 2 \cdot p$$

$$BIC = n \cdot \log\left(\frac{SSE}{n}\right) + p \cdot \log(n)$$

siendo: $n =$ número de observaciones y $p =$ número de parámetros

A la vista de que el segundo criterio es mucho más exigente que el primero podemos imaginar que el primero habrá dado lugar a la selección de un conjunto mayor de variables, y así es. Según AIC he seleccionado: **Pclass**, **Age**, **Title**, **CabNumber_bin**, **FamilySize** y **TickLetters_g**. Sin embargo, segun BIC basta con las variables: **Pclass** y **Title**.

Como la aplicación de estos criterios tiene cierta componente aleatoria he realizado la selección de forma recursiva probando distintas semillas (de generación de aleatorios) y me he quedado tan sólo las variables que eran comunes al 60% de los modelos.

De esta manera tengo ya cuatro conjuntos de variables (incluido el de TODAS) que me servirán para contrastar el desempeño de mis algoritmos, además cada conjunto contiene variables diferentes al resto (salvo el que las contiene todas).

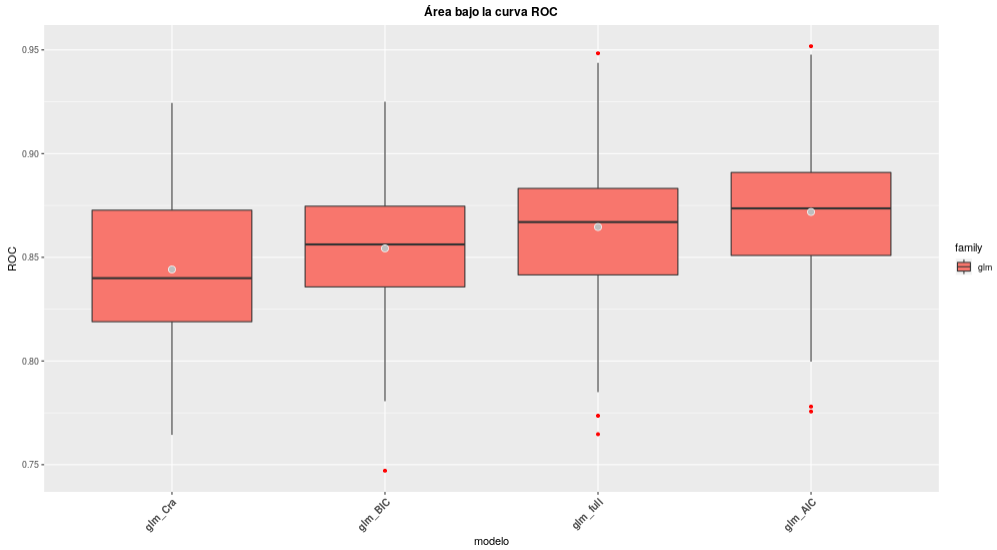
4 Modelos

A partir de ahora comienzo a generar modelos y a entrenarlos, por tanto los datos no deberían sufrir ninguna modificación más. Con el objetivo de satisfacer todos los puntos del enunciado no podré profundizar en cada modelo del proceso, pero si espero transmitir un sentido de la evolución a lo largo del proceso en la capacidad predictiva sobre el set.

4.1 Regresión Logística

Los primeros modelos que he entrenado han sido regresiones logísticas. Estando cada una asociada a la selección de variables explicada anteriormente, he realizado valización cruzada para todas ellas, obteniendo los siguientes datos:

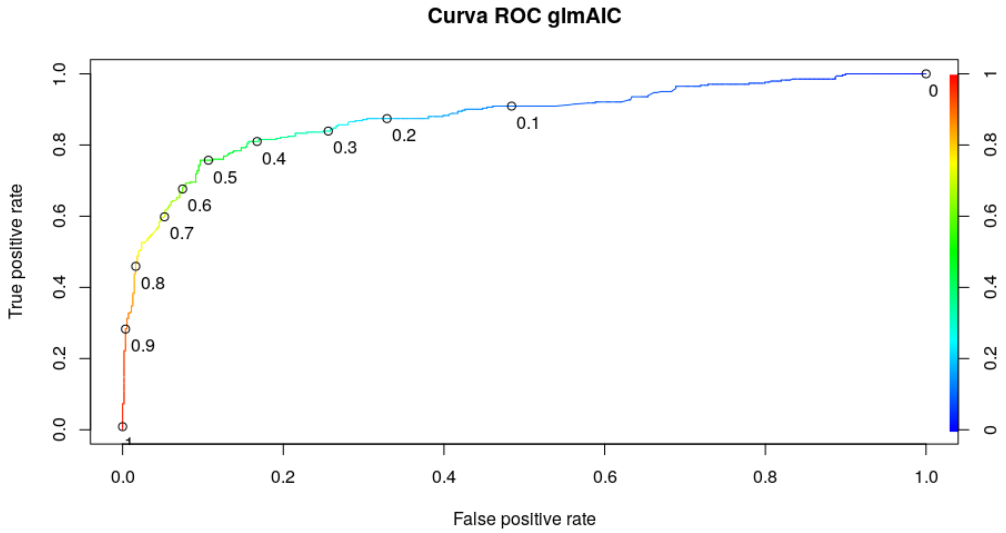
	mean_ROC	std_ROC	variables
glm_Cra	0.8441457	0.0339588	8
glm_BIC	0.8543086	0.0311712	7
glm_full	0.8646005	0.0323287	30
glm_AIC	0.8718738	0.0311339	14



El área bajo la curva ROC no está mal para ser los primeros modelos, pues supera el 84% en todos los casos.

Con el mejor modelo, que es glm_AIC he realizado una *submission* en Kaggle y he obtenido un: **79'425%** de aciertos, que nos debe servir como *baseline*. El comportamiento de la curva ROC para este modelo se puede observar en el gráfico. A partir de aquí debo mejorar.

Podemos ver también la curva ROC del modelo, que ayuda a hacerse una idea de en qué situación nos encontramos con respecto a un predicción promedio, que vendría representada por la diagonal. En este caso la curva crece muy rápidamente, para luego ir frenando gradualmente este crecimiento.



La representación de la curva ROC nos muestra como el modelo ya mejora bastante con respecto a la predicción por la media, la curva presenta un crecimiento muy rápido que se va atenuando, el objetivo es extender lo máximo posible ese crecimiento rápido con el fin de aproximarnos al punto (0,1) que es donde se ubicaría el modelo perfecto.

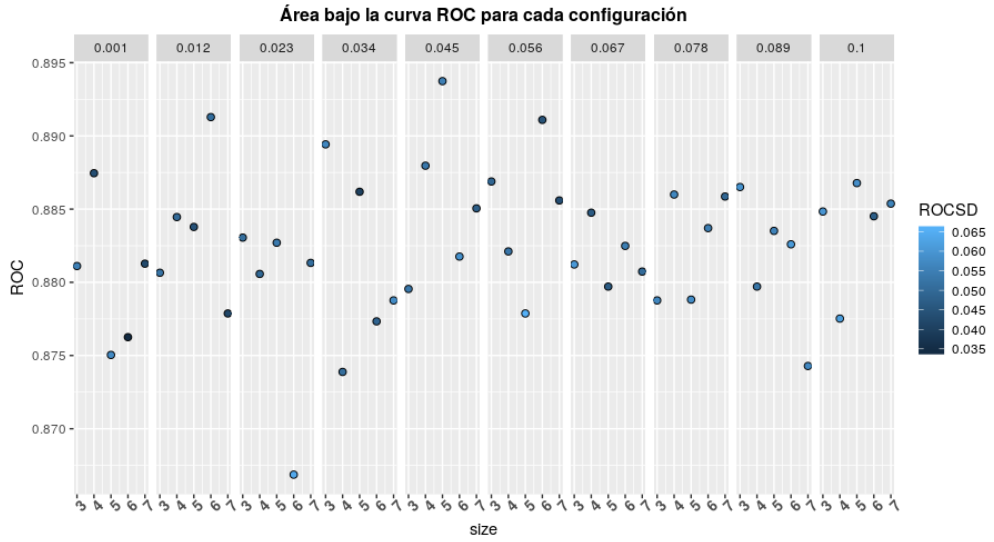
4.2 Perceptron Multicapa (MLP)

Voy a probar ahora a hacer predicciones con **redes neuronales** de una sóla capa oculta. Para ello voy a tener en cuenta el escalado de los datos y la ecuación fundamental a la hora de trabajar con estas redes, que es:

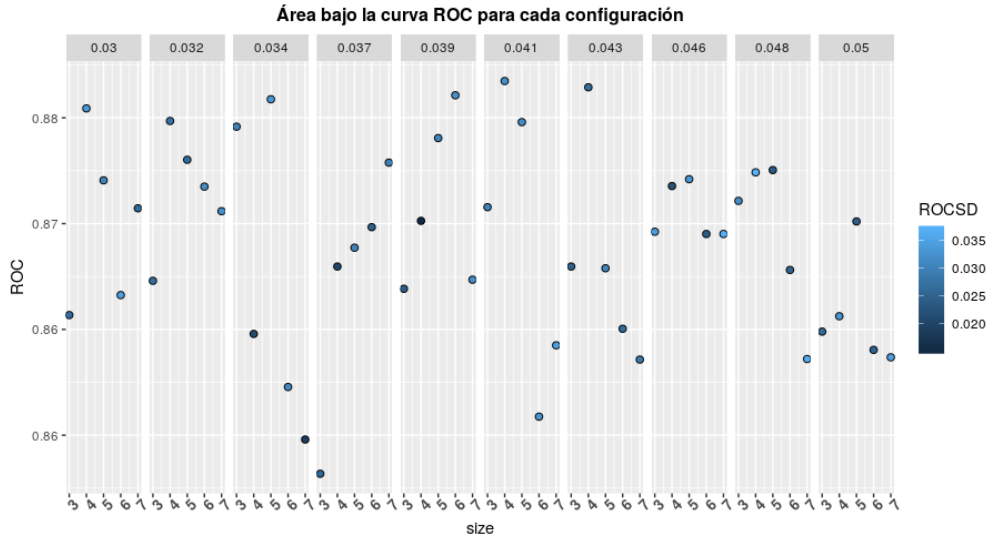
$$parámetros = h \cdot (k + 2) + 1$$

siendo: $h =$ nodos ocultos y $k =$ inputs

Según el manual de SAS, en problemas de clasificación conviene tener entre 5 y 25 observaciones por parámetro, en mi caso consideraré como mínimo 8, de esta forma para encontrar el límite superior al número de nodos ocultos de la red basta con calcular $\left(\frac{891\ obs.}{8\ obs./param} + 1\right) \cdot \frac{1}{14\ max.\ nodos\ input+2} \approx 7$. Por ello voy a entrenar redes con 3, 4, 5, 6 y 7 nodos ocultos. El ratio de aprendizaje lo haré variar entre 0.001 y 0.1, y me quedaré con los hiperparámetros que mejor resultado den para el modelo que emplea todas las variables, comprobando el desempeño de cada configuración haciendo *training - test* repetido, después haré validación cruzada para todos los modelos pero ya con esos parámetros fijos. Los resultados los he representado en un gráfico:



En él podemos ver que el mejor valor lo presenta la configuración 5 nodos y 0.045 de ratio de aprendizaje, sin embargo, este valor presenta una desviación alta, como los valores mayores aparecen concentrados en el intervalo 0.03 - 0.05 del ratio de aprendizaje he repetido el proceso restringido a este intervalo, obteniendo:



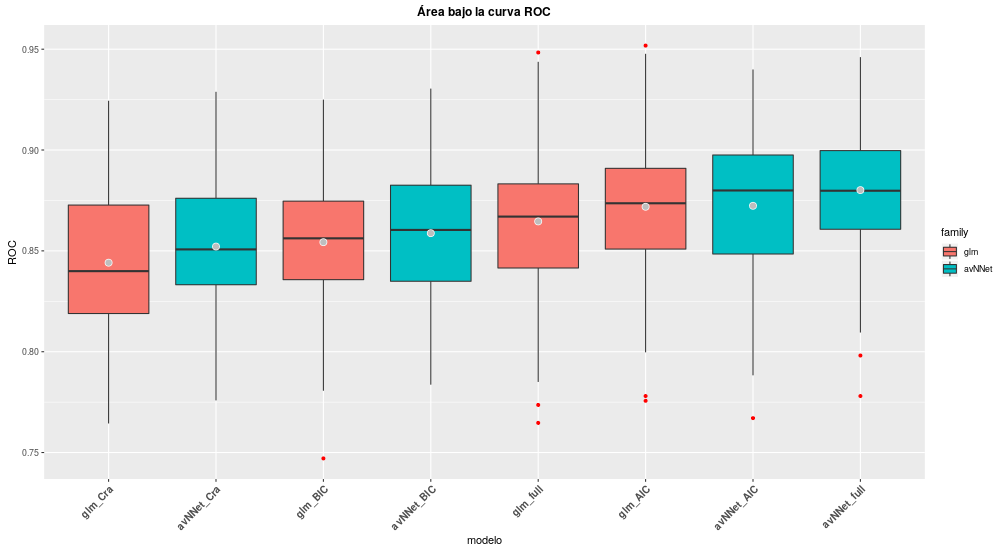
Donde ya podemos observar que la configuración óptima son 4 nodos ocultos y 0.041 ó 0.043 *learning rate*. Yo he empleado: 4 nodos ocultos y ratio de aprendizaje 0.041.

Con la citada configuración he realizado validación cruzada sobre las redes asociadas a cada uno de los conjuntos de variables declarados anteriormente, y he obtenido los siguientes resultados.

	mean_ROC	std_ROC
avNNet_Cra	0.8521404	0.0319674
avNNet_BIC	0.8587773	0.0325330
avNNet_AIC	0.8723537	0.0367709
avNNet_full	0.8801218	0.0338324

Podemos ver que el área bajo la curva ROC ha mejorado un poco con respecto a las regresiones logísticas, no obstante estamos lejos de poder hablar de una mejora significativa. Podemos visualizar la comparativa de los

modelos realizados hasta ahora en el gráfico siguiente:

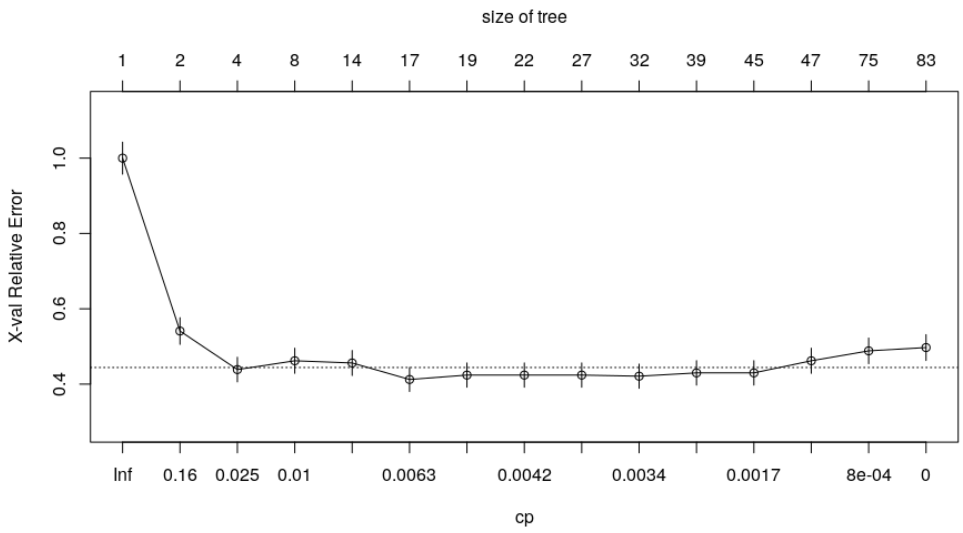


Y ver como el modelo que podemos suponer mejor hasta ahora es la red neuronal que emplea el set completo de datos, sin embargo al subir las predicciones a Kaggle hemos obtenido un: **77'511%**, luego no hemos mejorado.

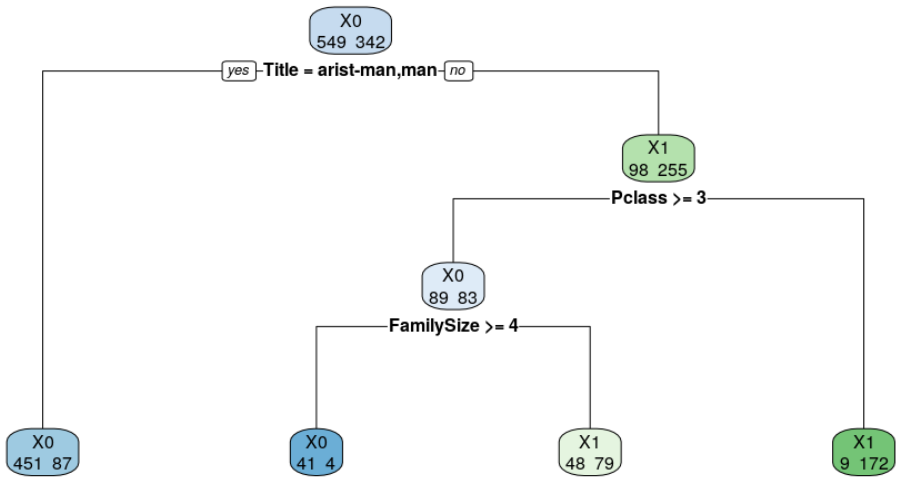
4.3 Árboles de decisión

Los árboles de decisión lo que hacen es fijar criterios que van segregando la muestra en conjuntos cada vez menores de observaciones que presentan mayor desproporción en cuanto a la variable objetivo. Además de ser muy sencillos de interpretar nos permiten realizar una nueva selección de variables, así que tras desarrollar este modelo tendremos un nuevo set de variables seleccionadas.

Lo primero que he hecho es ajustar los parámetros, de forma que tenga un modelo que de buena precisión pero no este sobreajustado. Tenemos que ajustar el parámetro `minbucket` que es el tamaño mínimo que puede tener la muestra en un nodo y `cp` que es un límite inferior para el descenso del el error relativo al añadir una nueva división entre el número de divisiones. El *minbucket* lo he sacado tanteando múltiples valores distintos y mirando las estructuras del árbol, hasta que finalmente 8 me ha parecido el valor óptimo. El `cp` lo he ajustado a la vista del siguiente gráfico:



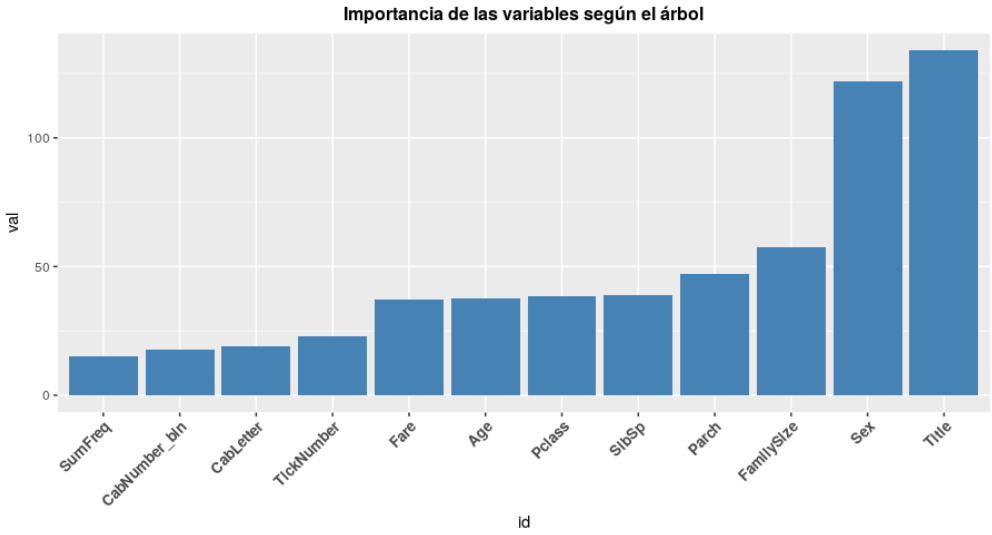
Donde podemos ver que valores inferiores a 0.025 no implican mejora. Con ambos parámetros fijados la estructura del árbol que ha quedado es la siguiente:



Podemos calcular la precisión : $100 \cdot \left(1 - \frac{87+4+48+9}{891}\right) = 83'389\%$ que es sobre el *set* de entrenamiento pero no está mal, maxime si tenemos en cuenta que solo ha considerado 3 reglas:

1. Es un adulto varón: Si = murío, No =
    2. Viajó en tercera clase: No = sobrevivió, Si =
        3. Su familia era de 4 miembros o más: Si = murió, No = sobrevivió

Podemos también ver ahora la importancia de las variables, únicamente ha empleado 3, sin embargo, el modelo asigna relevancia a todas ellas.

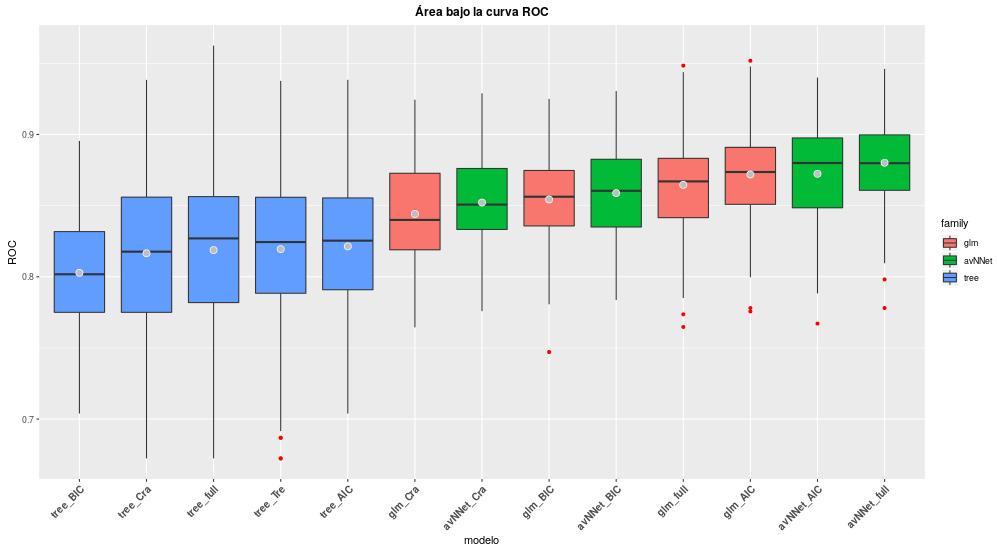


A la vista de esta ordenación he generado un quinto (y último) conjunto de variables, que emplearé junto al resto, y es: **Title**, **Sex**, **FamilySize**, **Parch**, **SibSp**, **Pclass**, **Age** y **Fare**.

Con estos cinco sets, he generado los árboles correspondientes y realizado validación cruzada para estimar su calidad. Los resultados obtenidos han sido los siguientes:

	mean_ROC	std_ROC
tree_BIC	0.8027292	0.0420070
tree_Cra	0.8164917	0.0549158
tree_full	0.8187442	0.0535437
tree_Tre	0.8194106	0.0521862
tree_AIC	0.8213425	0.0483928

El que presenta un mejor desempeño es el ajustado con las variables seleccionadas mediante el criterio AIC, pero los valores del área bajo la curva ROC son lo menores que hemos obtenido hasta ahora. Podemos mirar la comparativa en el gráfico con todos los modelos:



Y observar que todos los árboles quedan a la cola. De todas formas he seguido el procedimiento hasta el final,

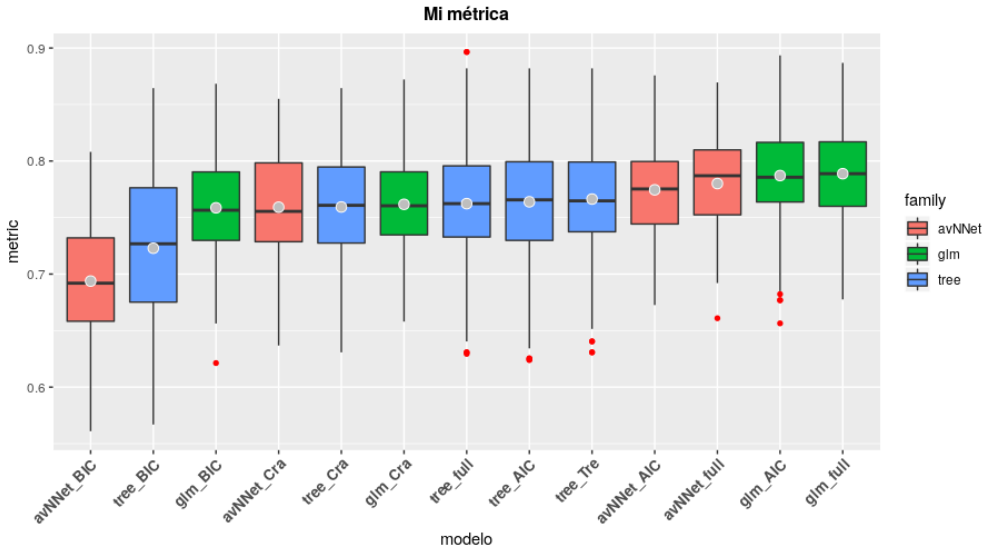
subiendo una predicción, y como cabía esperar el *score* ha empeorado: **75'598%**.

4.4 Nueva Estrategia

Mirando algunas tablas intermedias durante las ejecuciones de código, he observado que tal vez la métrica de la curva ROC tenía sentido para las regresiones logísticas, sin embargo, para el resto de algoritmos es una métrica posiblemente inadecuada (a mí, ya el uso del área bajo la curva y no de la distancia mínima de la curva al punto (0,1) en el plano (1 - Especificidad, Sensibilidad) me parece inadecuado). Por esto, y dado que estoy compitiendo, la métrica que voy a utilizar es una nueva métrica que he cocinado yo (sin tampoco devanarme los sesos), viendo que los modelos presentaban una especificidad (fallecidos acertados entre fallecidos reales) notablemente inferior a la sensibilidad (supervivientes acertados entre supervivientes reales) la he definido como:

$$metric = \frac{2}{3} \cdot Specificity + \frac{1}{3} \cdot Sensitivity$$

Equivalente a decir que los fallecimientos acertados cuentan doble. Si ordenamos los modelos que hemos realizado hasta ahora según esta nueva métrica quedan de la siguiente forma:

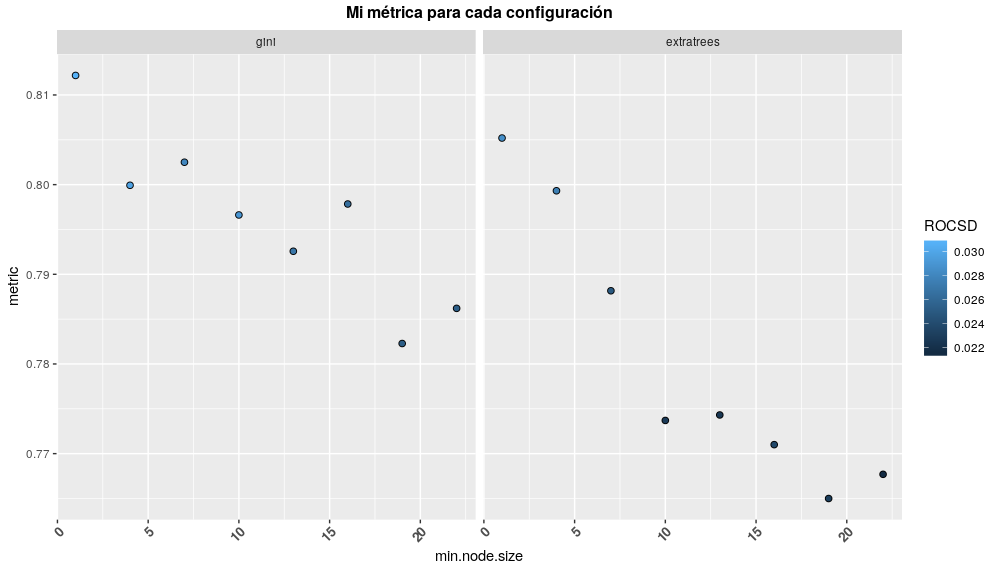


Y ya vemos como parece una mejor forma de seleccionar aquel que mejor predice, porque de los tres modelos que hemos usado para subir la predicción, el que mejor resultado nos ha dado aparece en cabeza. También vemos que el número de variables seleccionadas mediante el método BIC es muy reducido, y los modelos asociados presentan una capacidad predictora inferior al resto.

4.5 *Bagging* y *Random Forest*

Los modelos de árboles son muy inestables, esto quiere decir que presentan gran aleatoriedad, y según los parámetros que pongas, o el conjunto de datos que seleccionen al principio pueden surgir árboles diferentes, que evalúan las variables de maneras distintas. Para evitar este problema se desarrollaron los métodos de *random forest* y *bagging* (que es un caso de *random forest* particular). *Bagging* lo que hace es construir árboles con distintas submuestras de las observaciones, mientras que *Random Forest* consiste en tomar también submuestras de las variables. Tras construir los árboles la predicción final es fruto del promedio de las predicciones intermedias, es este sentido *bagging* es equivalente a la validación cruzada con reemplazamiento, pero en lugar de seleccionar el modelo que obtuvo mejor métrica se agregan los modelos y se predice mediante el promedio de todos ellos.

Aunque se nos dijo que para los árboles no era necesario escalar los datos, dado que los resultados del apartado anterior no han sido buenos, ahora si voy a emplear los datos escalados. Lo primero es usar *train - test* repetido para determinar que configuración de los árboles es la que mejor predice, obteniendo los siguientes resultados:



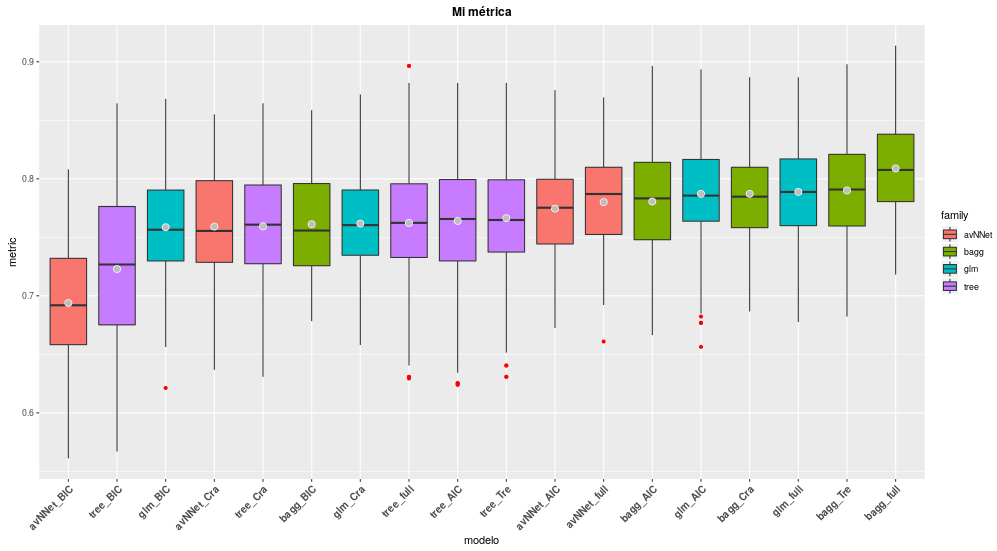
En la imagen se aprecia que claramente según disminuye el tamaño minímo que permitimos en un nodo aumenta el valor de mi métrica, siendo más aleatorio, pero mayor, el comportamiento que observamos si usamos como norma para seleccionar el punto de corte en los nodos el **coeficiente de Gini**:

$$I(Gini) = \left(1 - \sum_{i=1}^{k} \left(\frac{n_i}{n}\right)^2\right)$$

Usaré por lo tanto una configuración de 1 para el tamaño mínimo del conjunto en un nodo, y el coeficiente de Gini como norma para seleccionar el punto de corte. Además voy a emplear 300 árboles y un tamaño de la submuestra del 75% de la muestra total, por supuesto con reemplazamiento. Con esta configuración hago validación cruzada sobre los modelos asociados a los distintos *sets* de variables que hemos definido, y los resultados que he obtenido son:

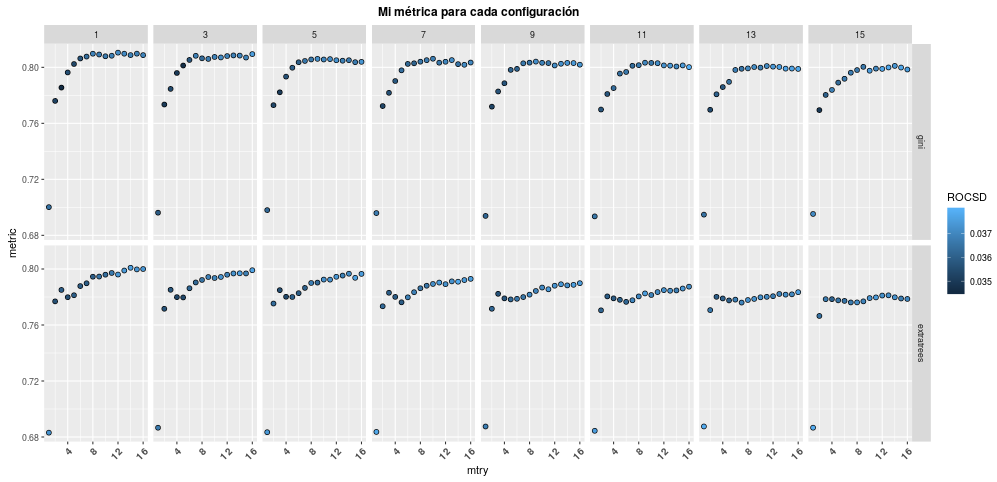
	mean_metric	std_metric
gbmBIC	0.7016488	0.0643314
gbmAIC	0.7644910	0.0442776
gbmTre	0.7814828	0.0368225
gbmCra	0.7940274	0.0399772
gbm_full	0.7972687	0.0401219

El inconveniente es que no podemos compararlos con los observados hasta ahora ya que hemos cambiado de métrica, sin embargo, si podemos verlo gráficamente, ya que el gráfico si está actualizado, con la métrica evaluada para cada modelo:

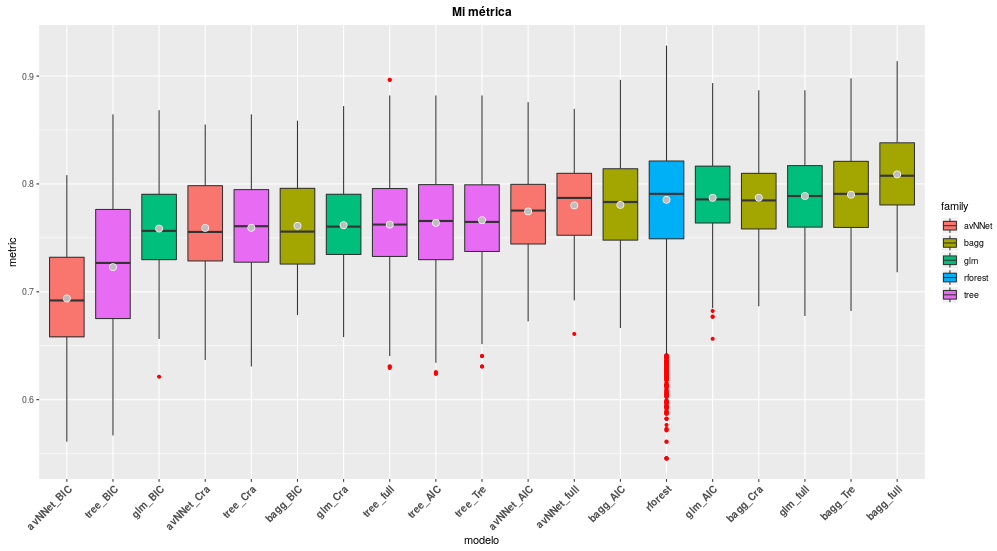


Vemos un resultado alentador, el algoritmo de *bagging* asociado al *set* completo de variables presenta un aumento sustantivo frente a la regresión logística con todas las variables, que era el mejor hasta ahora. Tanto es así que me dirijo ilusionado a realizar una nueva subida de predicciones a Kaggle y obtengo: **76'555%**. Es decir, una vez más (y van 4) no he mejorado los resultados del primer modelo.

Para el *random forest* únicamente voy a construir un modelo, ya que se van seleccionando subconjuntos de variables aleatoriamente en cada nodo, lo más sensato es entrenarlo con todas, y hacer estos conjuntos variar desde una variable hasta todas (si saliese que el mejor es con todas nos encontraríamos en el caso de *bagging*). Por lo tanto haré la validación cruzada directamente introduciendo el tamaño del conjunto de variable como un parámetro más del *grid*. Los resuiltados para el valor de mi métrica, evaluado para los modelos con distintos parámetros mediante validación cruzada son:



Siendo el modelo que ha obtenido un valor mayor aquel que usaba el coeficiente de Gini para determinar el punto de corte en el nodo, que tenía como tamaño mínimo permitido de la muestra en el nodo una única observación, y que sorteaba en cada nodo 8 variables (este valor lo vimos también como óptimo para el modelo de un único árbol). Esta es la configuración del modelo que presenta mejor desempeño, podemos comparar los resultados obtenidos con los de el resto de modelos:



Observamos que se coloca a tres cuartos de tabla, y con un varianza bastante elevada, de forma que no cabe hacerse ilusiones. La plataforma corrobora el augurio con un **78'468%**.

Como tenía muchas expectativas puestas en *random forest* ya que ha sido el algoritmo que hemos empleado en el modulo de **Aplicaciones del Big Data a la empresa** para obtener un *score* de ≈ 0.82 en este concurso, y tras leer este artículo sobre los distintos paquetes disponibles para hacer *random forest* en R, como hasta ahora había empleado **ranger**, me decidí a probar **cforest** ya que el artículo indica que mejora la precisión y reduce el sesgo, sin embargo, **cforest** no está disponible para la versión 3.6.0 de R (y no quería ponerme a intalar otra versión), así que mi gozo en un pozo.

4.6 *Gradient Boosting*

Al igual que con los algoritmos anteriores, lo primero que hay que hacer es ajustar los parámetros. Estos algoritmos permiten ajustar varios parámetros, los que yo voy a probar son:

- *Shrinkage*: Juega un papel similar al *learning rate* en las redes neuronales, ya que recoge información acerca de la velocidad del ajuste. Existe un compromiso entre este parámetro y el número de interaciones del algoritmo ya que si disminuimos el valor de este más iteraciones requerirá el algoritmo para converger y viceversa. Yo probaré los valores: 0.1, 0.05, 0.03, 0.01 y 0.001.
- Profundidad de las interacciones: Este parámetro recoge el orden de las interacciones entre variables que se incluirán en el modelo, hay que ser cuidadosos, ya que el aumento en el número de variables obedece al factorial, es decir:

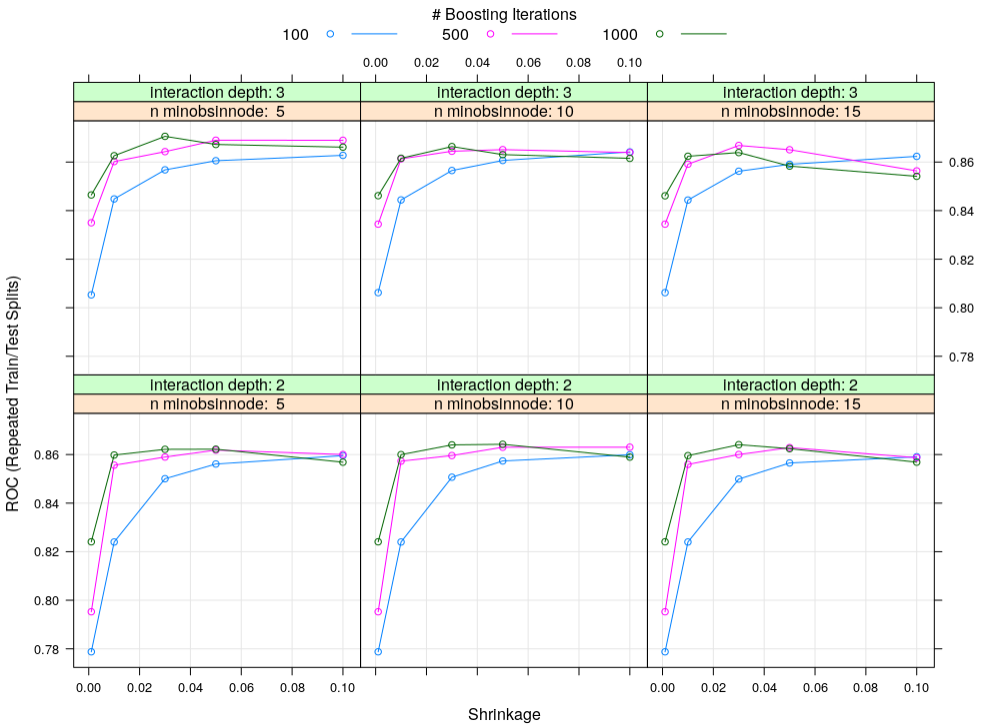
  $$\{\text{n. variables} = \text{N, interaction.deph} = \text{n}\} : \textit{variables nuevas} = N \cdot (N-1) \cdot (N-2) \cdot ... \cdot (N-n+1)$$

  Yo probaré únicamente los valores 2 y 3

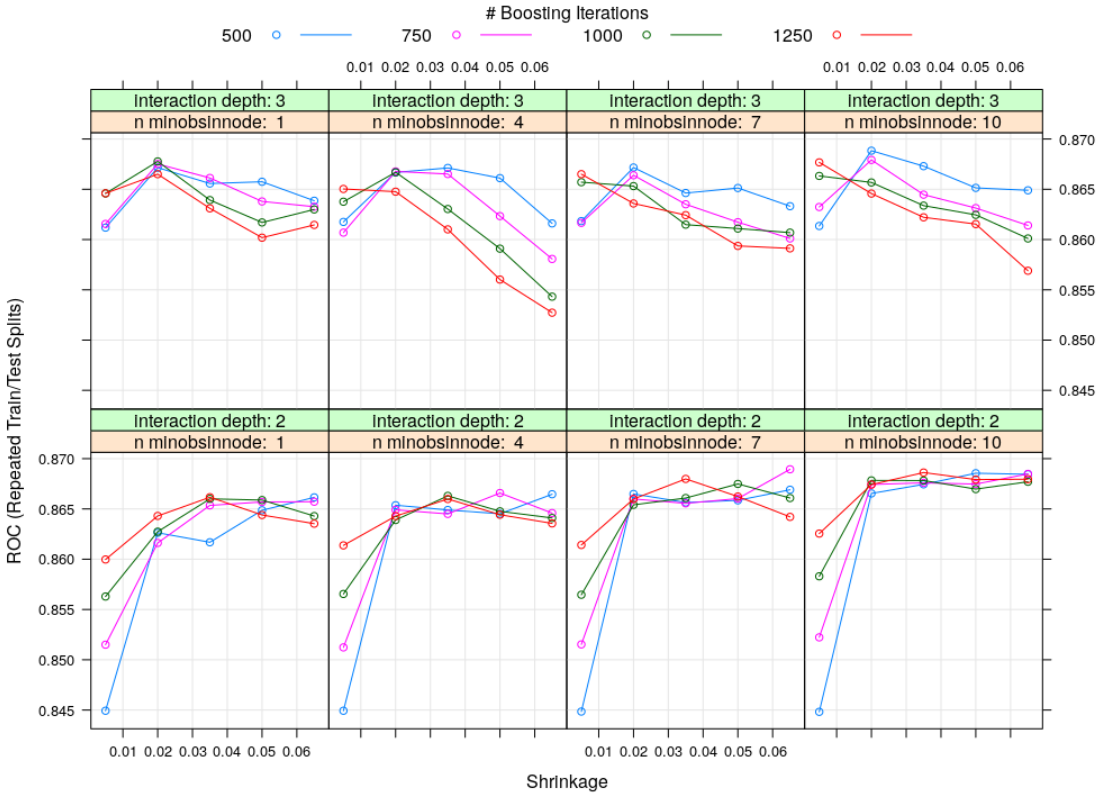
Los siguientes parámetros están relacionados con los árboles que se generan de forma sucesiva y son:

- Número de árboles: El número de árboles que se generarán en cada iteración. Probaré los valores: 100, 500 y 1000.
- Número mínimo de observaciones por nodo: Se autodefine y refiere a los nodos de los árboles. Yo he empleado: 5, 10 y 15.

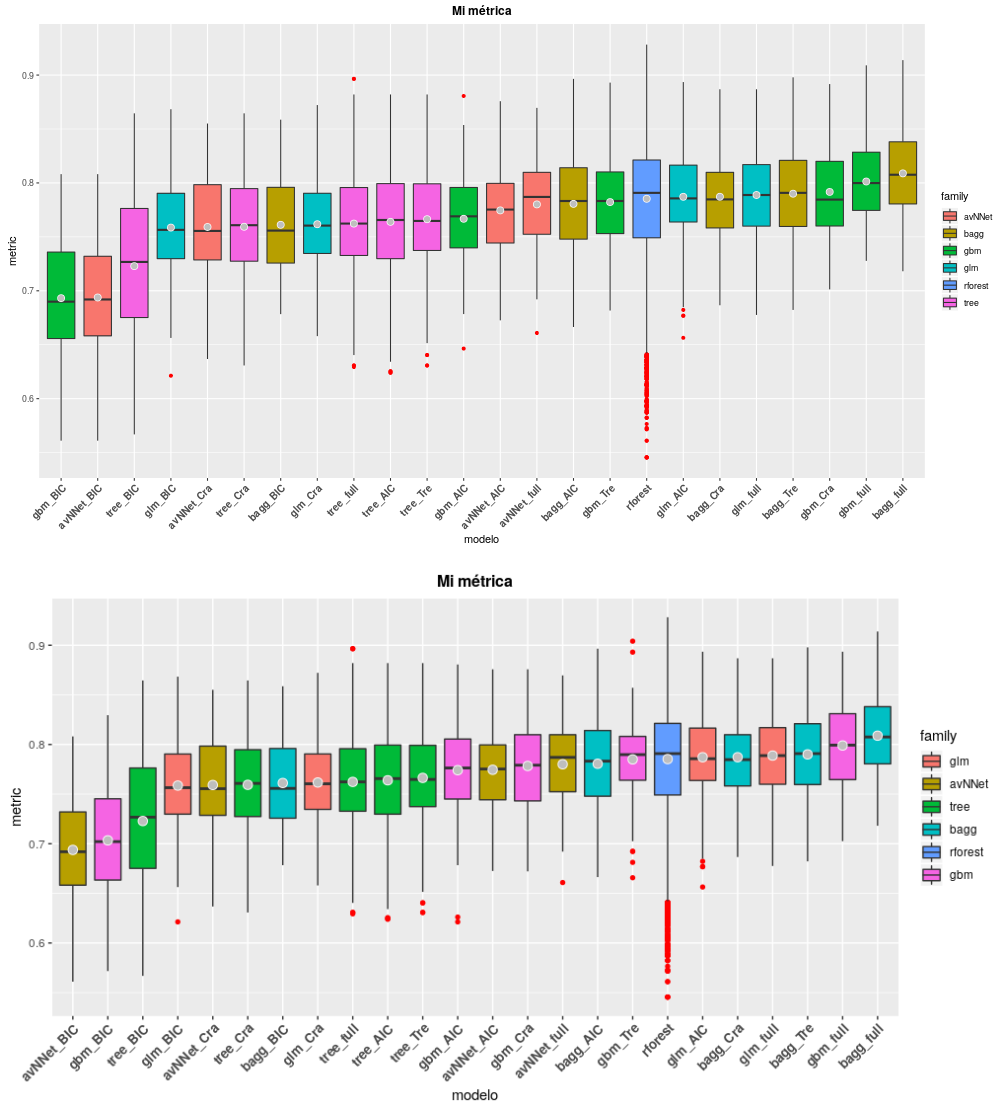
Los resultados de este ajuste se ven recogidos en la siguiente imagen:



Como a la vista de las curvas no tenía claro cual configuración era la óptima, repetí el proceso, reduciendo lo intervalos a las zonas en las que las curvas mejor se comportaban, y obtuve:



A la vista de los resultados elegí los parámetros que daban el modelo con mejor métrica, que eran, {*shrinkage* = 0.05, *interaction.deph* = 3, *n.trees* = 1250 y *n.minobsinnode* = 3}. Como estos parámetros dan una valor bastante bajo de la curva, ROC, sobretodo si lo comparamos con otras configuraciones, comencé a sospechar que tal vez el cambio de métrica había sido una mala idea y que el elegir los parámetros según este criterio me había impedido mejorar los resultados de los primeros modelos. Es por esto que elegí realizar la parte de comparativa con los modelos ya hechos y la subida de resultados tanto para los modelos con los parámetros seleccionados mediante mejor métrica como para aquellos seleccionados mediante mayor área bajo la curva ROC, que daba unos parámetros distintos, que eran: {*shrinkage* = 0.065, *interaction.deph* = 2 *n.trees* = 750 y *n.minobsinnode* = 7}. Para ambas configuraciones realicé los modelos pertinentes, y obtuve las siguientes gráficas comparativas:



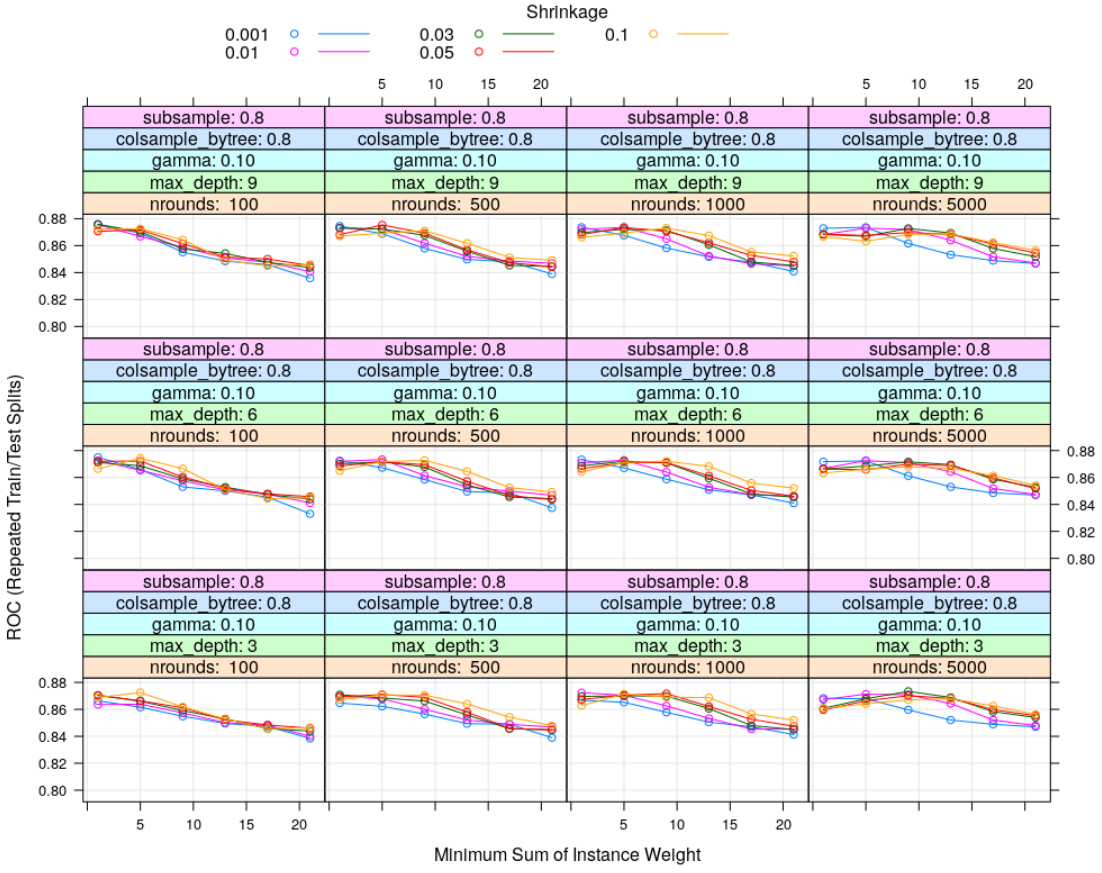
Aquí ya podemos ver que las diferencias no son llamativas. El algoritmo asociado a las variables seleccionadas con el criterio de Cramer decae bastante en el *ranking* y se reduce ligeramente la varianza al usar la configuración con mejor métrica. Además en ambos casos el mejor modelo es aquel entrenado sobre el *set* completo de variables. Tras hacer la subida de resultados descubrí que mis sospechas no podían ser más infundadas, el modelo con la configuración que resultaba en mejor métrica obtuvo: **76'076%** de forma que seguía sin mejorar mi marca, aunque no estaba lejos, sin embargo, el modelo con los parámetros que daban lugar a mayor área bajo la curva ROC obtuvo un **73'205%** lo que es la marca más baja registrada hasta ahora. Esto me disuadió de volver a la métrica de la curva ROC.

4.7 *XGBoost*

Se trata de un algoritmo desarrollado *ex professo* para ganar un concurso de Kaggle. Aunque no lo logró, sabemos que incorpora la regularización como proceso interno que participa de la optimización, de forma que se penaliza a los árboles según el valor de los pesos (α), la precisión de la predicción en cada hoja (λ) y el número de hojas (γ). La implementación del algoritmo en `caret` nos permite ajustar los parámetros:

- `eta`: Es lo que nosotros hemos llamado *learning rate* , es un factor (inferior a 1) que reduce la contribución de cada árbol.
- `gamma minimun`: Establece un límite inferior para la reducción de las función *loss* que se debe producir al añadir hojas al árbol.
- `min_child_weight`: Límite inferior para frenar que el árbol se siga particionando si la suma de los pesos no supera este valor.
- `subsample`: Porcentaje de observaciones seleccionadas aleatoriamente en cada iteración.
- `colsamble_bytree`: Porcentaje de variables seleccionadas aleatoriamente en cada iteración.

Dado que este es el penúltimo algoritmo que voy a usar y existe cierto apremio por mejorar los resultados he sido bastante ambicioso y he probado más de 6000 configuraciones (combinaciones de estos parámetros) distintas, tratando de asegurarme encontrar el mejor algoritmo. Hasta ahora mi postura respecto al ***machine learning***, sostenida por las lecciones del profesor Carlos Ortega, es que merece mucho más la pena invertir el tiempo en la manipulación de los datos (*data mining* y *feature engineering*) que en el tuneo de los parámetros de los modelos. Interesa comprobar que tu modelo no presenta una componente aleatoria fuerte usando métodos de validación repetida, pero más allá de eso no creo que el refinar en exceso el valor de los parámetros proporcione mejoras significativas. Dicho esto, dejo el ordenador computando, por si me equivocase. Los resultados para algunas de las configuraciones (tan solo vemos como varían según `shrinkage`, `max_deph` y `nrounds` pero se han hecho variar **todos** los parámetros) han sido:



Puedo también tabular las 15 configuraciones que han obtenido mejor valor para mi métrica:

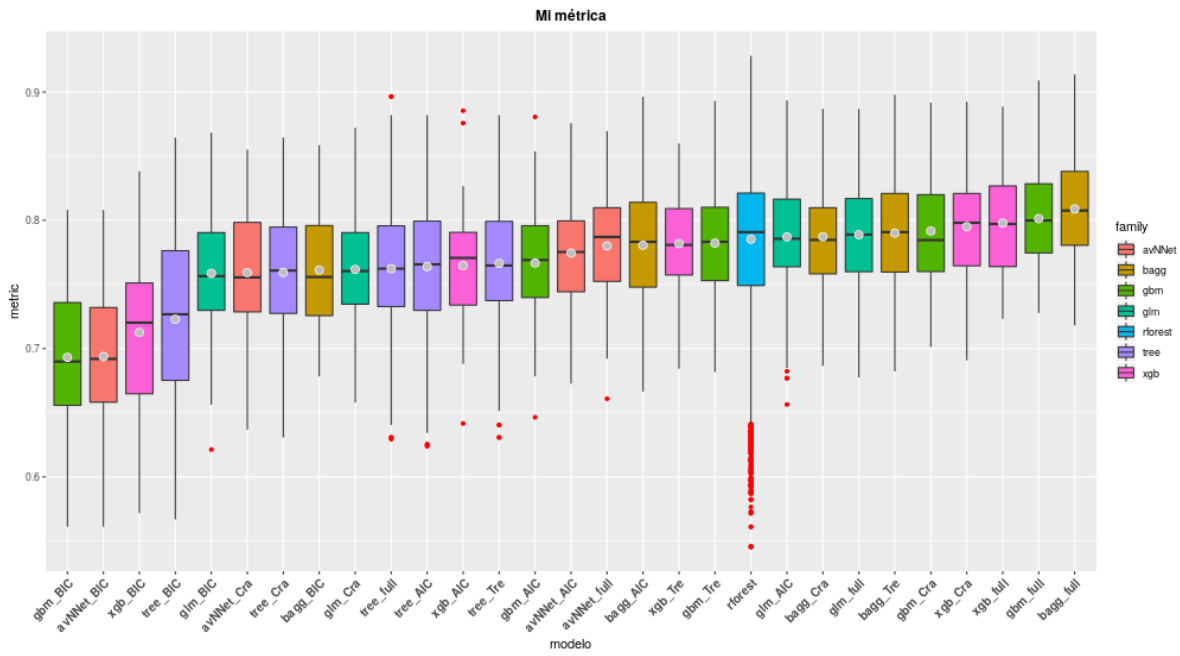
eta	max_d	gamma	colsam	min_w	subsam	nrnds	ROC	Sens	Spec	metric
0.03	6	0.05	0.8	1	0.8	5000	0.8661630	0.8660550	0.7617647	0.7965282
0.10	6	0.10	0.6	5	0.8	1000	0.8697787	0.8605505	0.7647059	0.7966541
0.05	9	0.10	0.8	1	0.8	500	0.8681597	0.8733945	0.7588235	0.7970139
0.05	6	0.00	0.8	1	0.8	1000	0.8675391	0.8733945	0.7588235	0.7970139
0.10	6	0.05	0.6	1	0.5	500	0.8651376	0.8678899	0.7617647	0.7971398
0.10	3	0.00	0.6	1	0.5	1000	0.8606584	0.8678899	0.7617647	0.7971398
0.03	9	0.05	0.6	1	0.8	1000	0.8696438	0.8807339	0.7558824	0.7974996
0.10	9	0.00	0.6	1	0.5	500	0.8635726	0.8752294	0.7588235	0.7976255
0.05	6	0.10	0.8	1	0.5	1000	0.8698057	0.8752294	0.7588235	0.7976255
0.05	6	0.05	0.8	1	0.8	1000	0.8655693	0.8697248	0.7617647	0.7977514
0.10	6	0.05	0.8	1	0.8	500	0.8680788	0.8770642	0.7588235	0.7982371
0.01	6	0.10	0.8	1	0.8	5000	0.8668106	0.8770642	0.7588235	0.7982371
0.03	6	0.00	0.6	1	0.8	5000	0.8677550	0.8715596	0.7617647	0.7983630
0.10	3	0.00	0.4	1	0.5	1000	0.8654884	0.8733945	0.7617647	0.7989746
0.10	3	0.00	0.4	1	0.5	500	0.8657312	0.8807339	0.7588235	0.7994603

A la vista de los resultados he empleado la configuración que mejor valor para mi métrica ha obtenido, y con ella he entrenado los modelos asociados a los conjuntos de variables que venimos manejando desde el principio.

Hecho esto los resultados obtenidos son los siguientes:

	mean_metric	std_metric
xgb_BIC	0.7125509	0.0643224
xgb_AIC	0.7647235	0.0425390
xgb_Tre	0.7818087	0.0406339
xgb_Cra	0.7950733	0.0431820
xgb_full	0.7978405	0.0426614

Por enésima vez las variables seleccionadas según el criterio BIC implican una pérdida significativa de información, mientras que los árboles, el criterio basado en la V de Cramer y el conjunto completo (que es el que mejor resultados da) vienen comportándose de forma similar. Estos modelos comparados con los que he realizado hasta ahora, presentan un buen desempeño, como muestra el gráfico.



Lo que nos permite pensar en ese bendito aumento del *score* que no se ha producido desde el primer modelo. Tras subir los resultados a Kaggle la frustración se vuelve a hacer patente ya que obtenemos un **77'03%**, un resultado de nuevo inferior a la mejor marca.

4.8 Máquinas de vectores de soporte (SVM)

Las máquinas de vectores lo que hacen es tratar de encontrar el hiperplano que mejor separa las categorías en el espacio de las variables $\left(\mathbb{R}^N\right)$, donde mejor quiere decir que la distancia entre las obsevaciones de distinta clase y el hiperplano es la mayor posible. Como la separación perfecta no suele existir, permitimos que algunas observaciones queden mal clasificadas introduciendo un error ($\xi \geq 0$). De esta forma lo que buuscamos es:

$$\arg\max \left[||\bar{w}|| + C \cdot \sum_{i=1}^{N} \xi_i \right]$$

Tal que el plano es: $\bar{w} \cdot \bar{x} = b$

Un concepto novedoso y que permite que el algoritmo rinda mejor en las ocasiones en que la separación entre clases no es lineal, es trabajar con dimensiones adicionales representadas por transformaciones de las variables, ya que los puntos sobre el espacio construido con estas dimensiones adicionales si pueden admitir una separación lineal. Dado que las posibles transformaciones que admiten las variables son infinitas, tanto más (entiéndase la expresión) cuantas más variables tengamos, se emplea lo que se conoce como ***Truco Kernel***. Que nos permite aumentar la dimensionalidad del espacio sin pasar, *de facto* por la inclusión de variables nuevas.

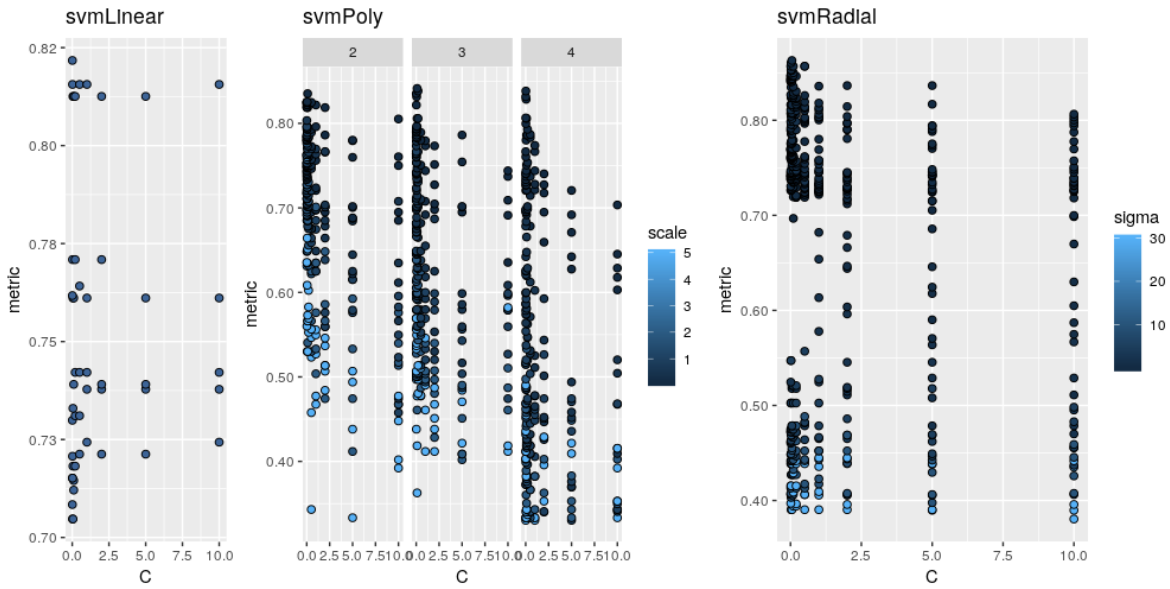
$$\text{Siendo: } \bar{w} = \sum_i \alpha_i y_i \bar{x}_i$$

$$\mathcal{L}(\alpha) = \sum_i \alpha_i + \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathrm{K}(\bar{x}_i \cdot \bar{x}_j)$$

Siendo $\mathcal{L}$ el lagrangiano a maximizar y K la función *kernel*

Generalmente los parámetros que solemos manipular son `C` que muestra una fuerte dependencia de los datos y `K` la función kernel. Dependiendo de la función kernel que usemos disponemos de más o menos parámetros

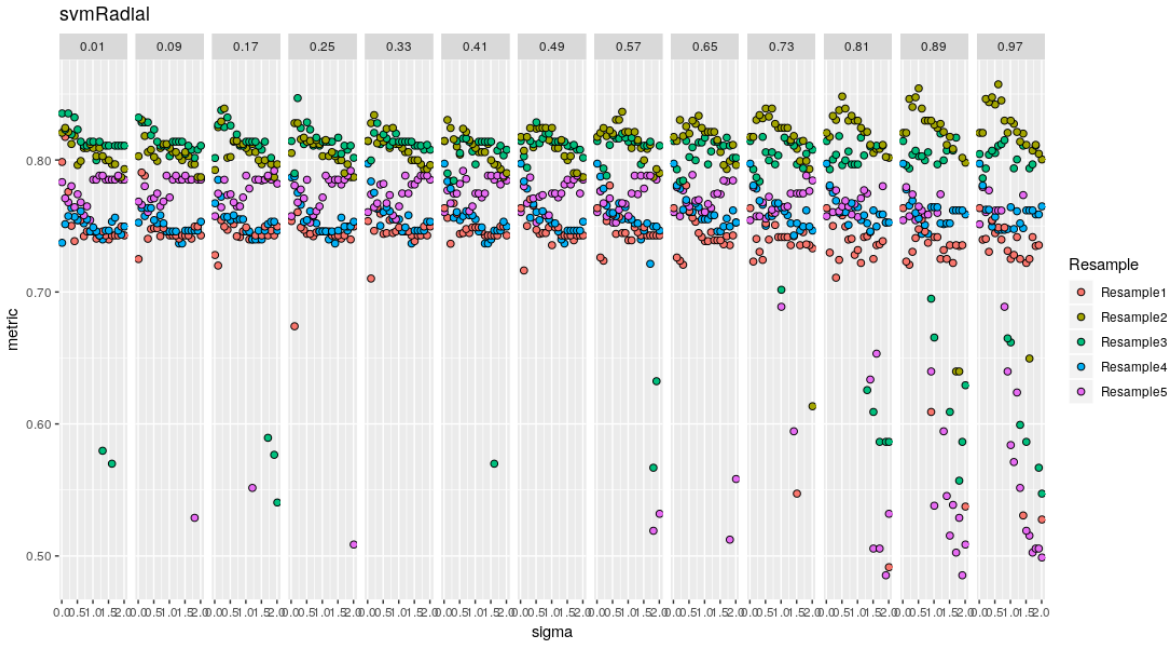
para ajustar, así que yo voy a entrenar modelos con las 3 funciones kernel disponibles, a representar su desempeño, y en base a esto eligiré cual de ellos usaré para generar mis modelos con distintas variables. Los resultados de la etapa de ajuste de parámetros los he recogido en el siguiente gráfico:



Dado que hay muchos puntos, podemos observar los parámetros asociados a las 15 configuraciones que obtuvieron mejor métrica.

ROC	Sens	Spec	C	Resample	metric	method	degree	scale	sigma
0.8881543	0.6880734	0.9264706	0.01	Resample4	0.8470049	svmRadial	0	0	0.20
0.8878845	0.6880734	0.9264706	0.02	Resample4	0.8470049	svmRadial	0	0	0.20
0.8880194	0.6880734	0.9264706	0.05	Resample4	0.8470049	svmRadial	0	0	0.20
0.8874798	0.6880734	0.9264706	0.10	Resample4	0.8470049	svmRadial	0	0	0.20
0.8774960	0.7247706	0.9117647	0.02	Resample4	0.8494334	svmRadial	0	0	0.50
0.8714247	0.6972477	0.9264706	0.10	Resample4	0.8500630	svmRadial	0	0	2.00
0.8908527	0.7339450	0.9117647	0.02	Resample4	0.8524915	svmRadial	0	0	0.05
0.8909876	0.7339450	0.9117647	0.05	Resample4	0.8524915	svmRadial	0	0	0.05
0.8765515	0.7064220	0.9264706	0.02	Resample4	0.8531211	svmRadial	0	0	1.00
0.8765515	0.6880734	0.9411765	0.01	Resample4	0.8568088	svmRadial	0	0	1.00
0.8765515	0.6880734	0.9411765	0.10	Resample4	0.8568088	svmRadial	0	0	1.00
0.8783055	0.6880734	0.9411765	0.50	Resample4	0.8568088	svmRadial	0	0	1.00
0.8712898	0.6880734	0.9411765	0.50	Resample4	0.8568088	svmRadial	0	0	2.00
0.8710200	0.6697248	0.9558824	0.02	Resample4	0.8604965	svmRadial	0	0	2.00
0.8764166	0.7064220	0.9411765	0.05	Resample4	0.8629250	svmRadial	0	0	1.00

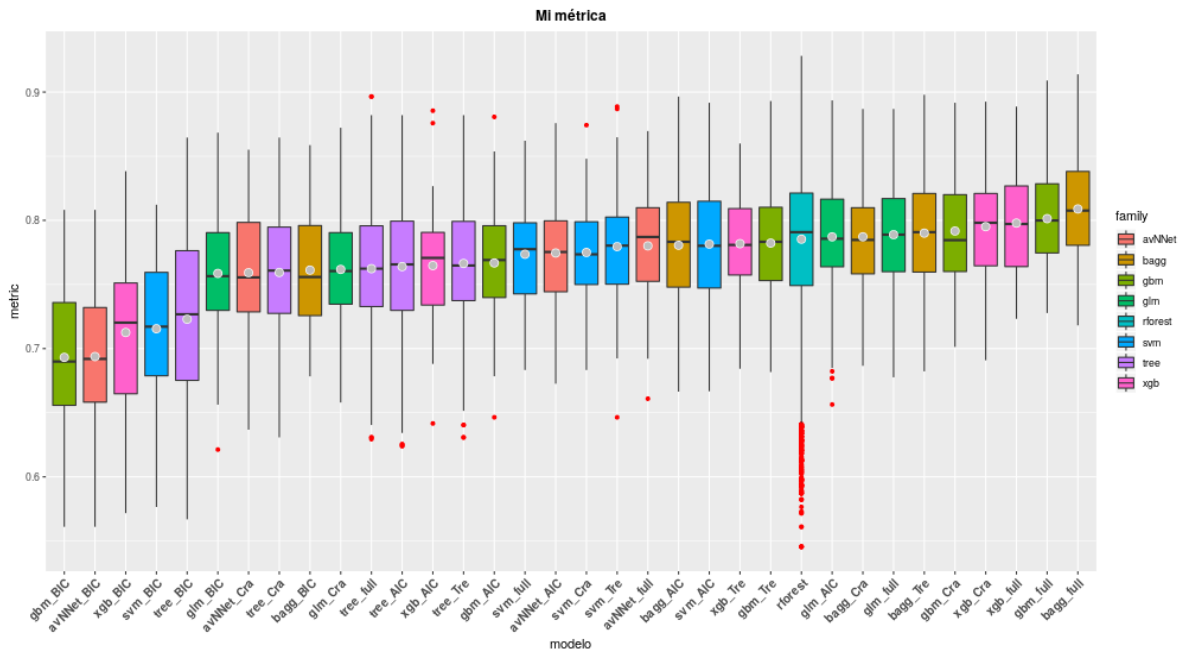
A la vista de los resultados entendí que la función kernel que debia usar era `svmRadial`, no obstante los valores de sus parámetros `C` y `sigma` no estaban claros, así que refine el tuning entre los valores que vemos en la tabla, y obtuve los siguientes resultados.



Estoy representando la metrica obtenida en función del parámetro `sigma` para cada valor de `C`, separando con colores cada repetición *train-test*. Esta imagen me sirve para justificar el no haber comprobado en todo el trabajo que los modelos eran robustos frente a la inicialización con diferentes semillas. Esto no tiene sentido cuando vamos a repetir el procedimiento de validación ya que la semilla de selección de la muestra va a ser distinta, de forma que la única susceptibilidad del modelo se daría si justo la semilla fuera siendo la idónea para la partición que se empleará en esa repetición, considerar esto una inconsistencia, más aún cuando el procimiento de validación es validación cruzada (el que empleo sobre los modelos a comparar), es un completo disparate. Por ello no he incluido un *loop* sobre diferentes semillas.

Dicho esto podemos observar que en este caso la partición asociada a la primera repetición resulta bastante más dificil de predecir que cualquier otra, sobre todo para valores altos de `sigma`. Podemos ver como todos los gráficos presentan algún punto completamente desviado inferiormente, salvo para los valores de `C` igual a 0.33 y 0.49. Por lo tanto he considerado óptimo `C = 0.4`. Los valores de `sigma` que dan mejores resultados los encontramos entre 0 y 0.5, por lo que he usado `sigma = 0.2`. Con esta configuración he entrenado mis modelos.

Desde luego en los apuntes dice que los datos que son bien ajustados por modelos de regresión logística suelen ser ajustados mejor por modelos de máquinas de vectores, así que veamos como queda la imagen comparativa de los modelos:



Vemos que el que mejor queda se coloca en décimo tercer lugar, sin presentar un desempeño en absoluto impresionante. No obstante como ya es tradición subo las predicciones pertinentes y obtengo como resultado un **78'947%** que no mejora mi mejor marca pero si es un valor superior al de la mayoría de los modelos empleados.

4.9 Conclusiones

No puedo decir que no haya sido duro ver que varios días, 17 páginas y 28 modelos después no he sido capaz de mejorar las predicciones realizadas por la regresión logística. No obstante todavía queda el ensamblado, y aunque los modelos realizados hasta ahora hayan dado un resultado mediocre, tal vez cooperando podamos escalar posiciones en el *leaderboard*.

5 Ensamblado

Lo primero que voy a hacer es comprobar si mis modelos realizan predicciones diferentes sobre el set de datos test, de forma que si me estoy moviendo en torno a un **78%** de precisión pueda saber si hay suficientes observaciones que cambian de valor según el modelo como para pensar en una mejora significativa de la predicción. Para ello primero voy a agregar a las predicciones 2 valores promedio:

- Promedio simple: Como su nombre indica se trata de un promedio equiponderando todos los modelos. Esta agregación la he realizado tanto en las predicciones binarias como en las probabilísticas, considerando, en las binarias, que los valores inferiores a 0.5 no sobrevivieron y los superiores si.
- Promedio ponderado: Para elegir el valor de la ponderación primero he ordenado los modelos según el valor que obtuvieron en promedio para mi métrica haciendo validación cruzada. Como el rango era relativamente pequeño ([0.7, 0.8]) si basaba la ponderación directamente en estos valores no iba a haber grandes diferencias con el promedio simple y tampoco podía escalarlo a [0, 1] por que la diferencia no era suficiente como para ignorar las predicciones del peor modelo. Así es que escale el rango linealmente al intervalo [0.3, 0.7] y basé en estos valores mi ponderación. Traas calcular el promedio ponderado ocurría que modelos con alta ponderación presentaban también probabilidades altas con lo que aparecieron valores mayores que 1, mientras que el menor era 0.05. Ahora si reescale este rango linealmente a [0, 1]

y me quedé estas probabilidades. Para las predicciones binarias simplemente ponderé y usé el mismo criterio que con el promedio simple.

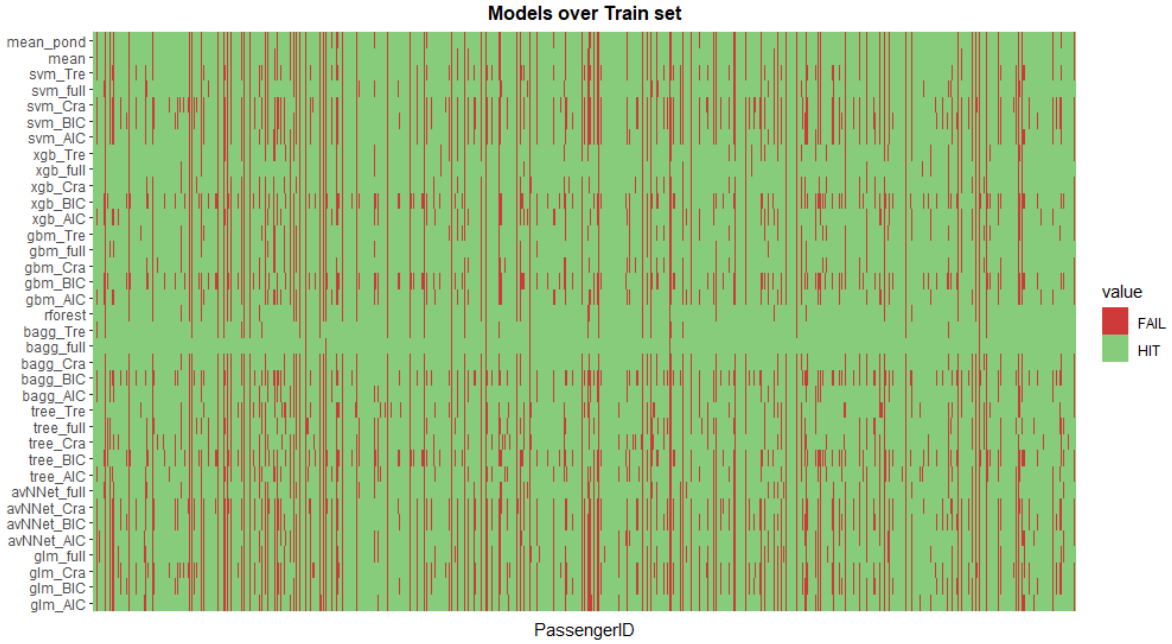
Añadidas estas dos nuevas predicciones a las propias de cada modelo me dispuse a contar las observaciones en las que al menos 1 modelo difería del resto, y encontre que eran un 36.48% (325) para el *set* de entrenamiento y un 36.84% (154) para el de test. El hecho de que ambas presentasen porcentajes similares reforzó la idea de que el *set* de entrenamiento es representativo del *set* de test.

Como los promedios propuestos ya son formas, aunque un poco *naive*, de ensamblado, los empleé para subir nuevas predicciones, obteniendo los resultados: **78'947%** para el promedio simple y **79'425** para el promedio ponderado. Son bastante buenas noticias porque el promedio ponderado iguala el *score* obtenido por la **regresión logística** con las variables seleccionadas mediante el criterio **AIC**, que era el mejor valor obtenido hasta ahora.

5.1 *Clustering* de observaciones

Desde que vimos el ensamblado vengo pensando que una buena forma de mejorar el *score* tiene que ser usar clustering sobre los datos, detectar que modelos son los que mejor predicen para cada conjunto, y emplear únicamente esos modelos en él. No tengo ningún argumento para justificar esta idea más que la propia intuición, pero me parece una forma elegante de finalizar el trabajo.

El *clustering* lo he hecho basándome en las variables originales más las predicciones de los modelos. Como he realizado 34 modelos, más las predicciones promedio y promedio ponderado he añadido 36 variables al *set*. Es más que razonable pensar que si queremos encontrar que modelo predice mejor en cada grupo añadir sus predicciones solo va a favorecer que encontremos esta agrupación. Lo primero que podemos hacer es ver en un mapa de calor los aciertos y fallos de los modelos, para orientarnos.



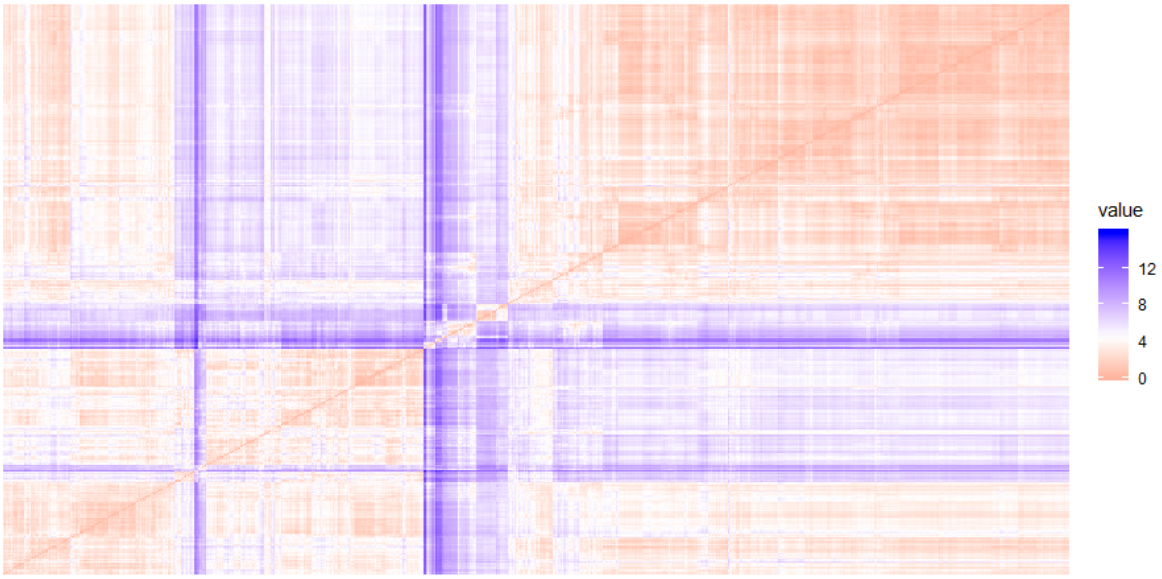
Aquí vemos que los modelos *bagging* presentan una precisión sobre el *set* de entrenamiento muy buena, lo que es coherente con la posición que presentaban en los gráficos. También vemos que en general los individuos en los que se cometen errores son comunes a todos los modelos. No obstante podemos observar también que hay unos modelos que son bastante mejores que otros. De hecho, siendo un poco puntillosos, encontramos que solo hay 5 individuos en los que fallen todos los modelos (los 5 en los que falla ***bagg_full***, 99,994% de precisión, muy sobreajustado ¿no?). Podemos mirar algunas variables de forma pormenorizada:

Sur.	Cl	Sex	Age	SSp	Pch	Fare	Emb	CabL	CabB	TickN	Title	SnFr
1	3	female	19.00	1	0	7.8542	S	U	0	350046	woman	1
0	3	male	24.00	0	0	7.0500	S	U	0	3101311	man	2
1	1	female	36.00	0	0	135.6333	C	C	1	17760	woman	1
1	1	female	22.00	0	1	55.0000	S	E	1	113505	woman	1
0	3	female	31.00	0	0	7.8542	S	U	0	350407	woman	2
1	1	female	44.00	0	1	57.9792	C	B	1	111361	woman	2
1	3	male	0.42	0	1	8.5167	C	U	0	2625	boy	1

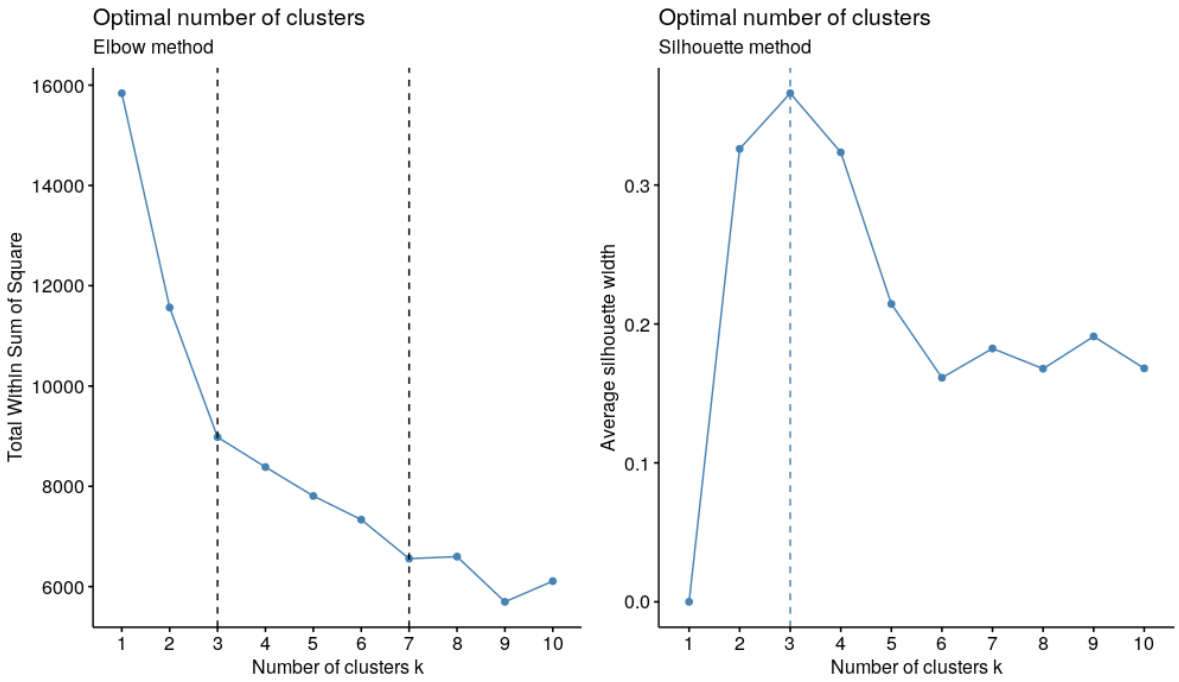
Pero, salvo que algo se me esté pasando, no parece existir un patrón claro entre ellos.

Ahora hay que seleccionar el número óptimo de clústers. Como he hecho la agrupación mediante la distancia euclídea sobre los datos (con predicciones **probabilísticas** incluidas) escalados, podemos antes ver la distribución de distancias:

Distancias entre los individuos



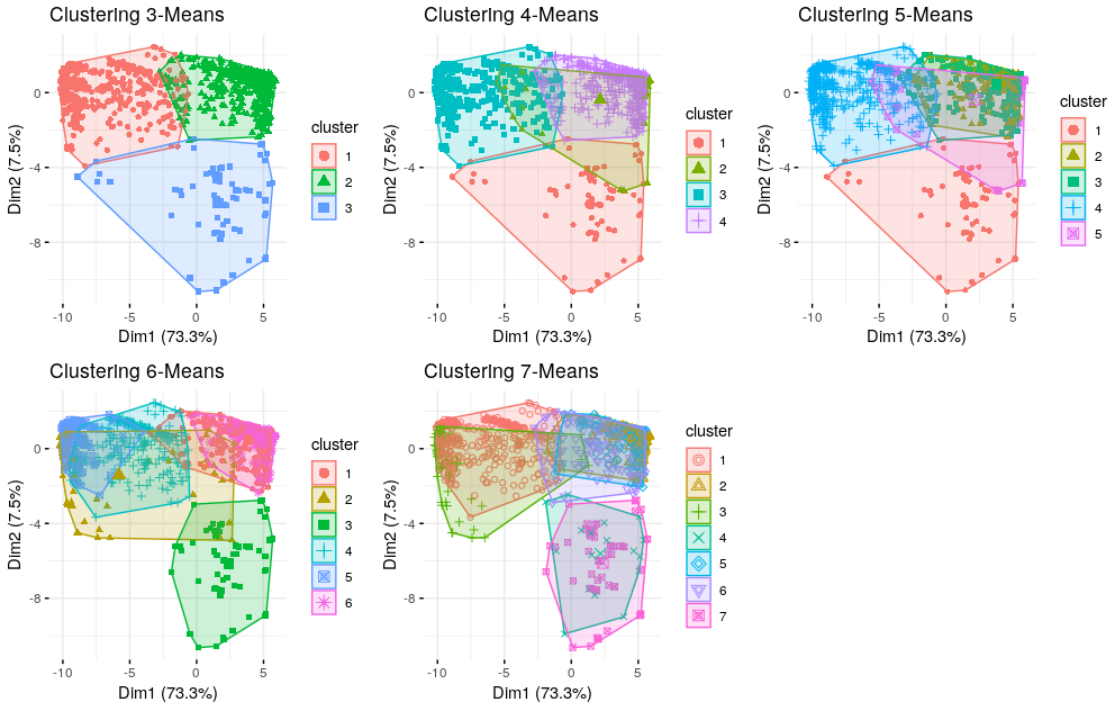
Ya vemos que la tonalidad delimita múltiples regiones. Si usamos ahora los criterios ***elbow*** (minimiza la variabilidad total intraclúster) y ***silhouette*** (maximiza la compacticidad de los clústers) para seleccionar el número óptimo de clústers podemos ver que según el primero el número óptimo está en 3 y 7, y según el segundo el mejor es marcadamente 3.



Con estos valores como orientación voy a realizar el clustering mediante próximos vecinos y mediante jerarquía, para ver si las agrupaciones son consistentes y acabar seleccionando la más convincente para proceder a la división de los datos.

5.1.1 *K-Means*

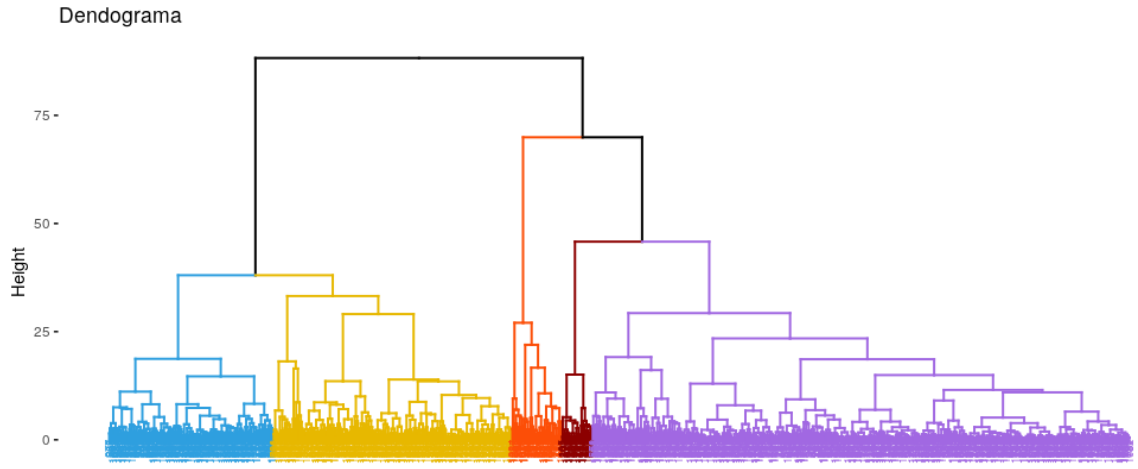
Al agrupar las observaciones en 3, 4, 5, 6 y 7 conjuntos considerando las distancias entre los datos y representarlas sobre el plano de las dos primeras componentes principales obtengo los siguientes gráficos:



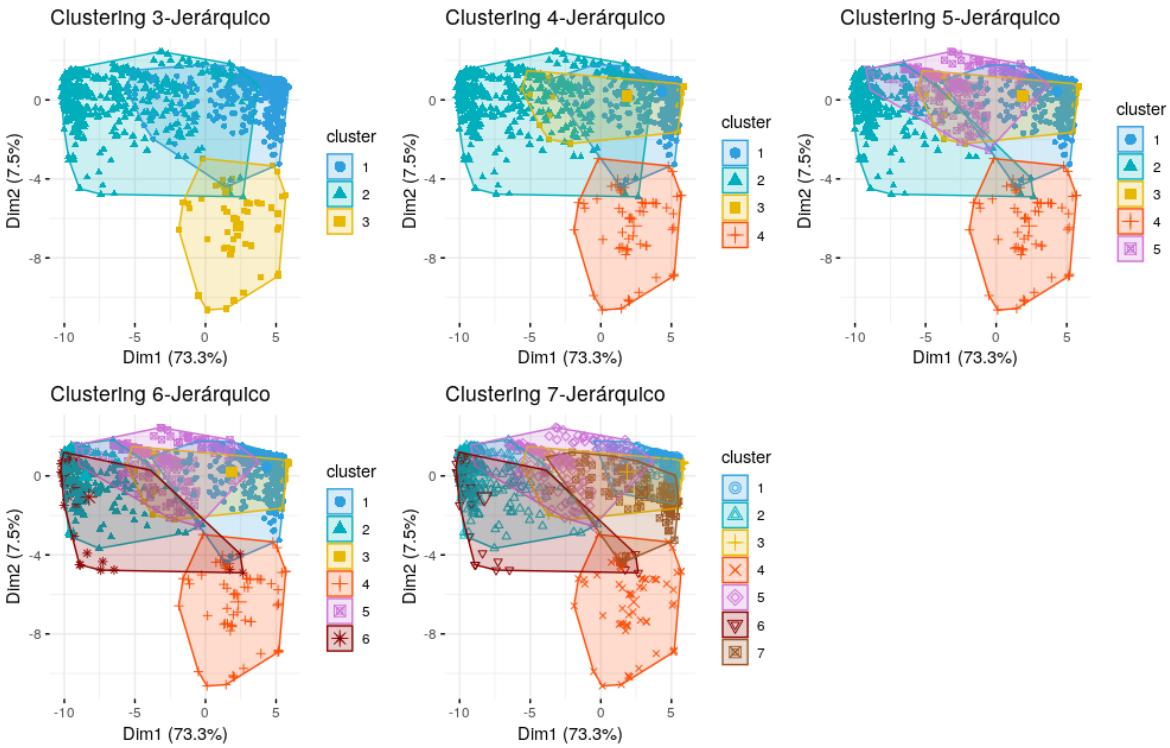
Lo primero que hay que señalar es que las dos componentes principales explican algo más del 80% de la variabilidad del conjunto de datos, para lograr un 90% o un 95%, que suele ser lo deseado debemos considerar más componentes, no obstante, eso es harina de otro costal. De momento me contento con notar que la agrupación en 3 conjuntos los "independiza" prácticamente del todo, mientras que en la agrupación en 7 hay conjuntos totalmente solapados con otros. Por otra parte vemos que la agrupación en 4 y 5 conjuntos insiste en diferenciar individuos situados en la esquina superior derecha. La agrupación en 6 muestra diferencias notables con la agrupación en 5, ya que aparece un conjunto que aglomera observaciones por la parte izquierda y central superiores. A la vista de las imágenes la agrupación óptima la encuentro en 5 conjuntos, ahora hay que mirar el *clustering* jerárquico.

5.1.2 Jerárquico

Lo primero que nos viene a la cabeza al escuchar *clustering* jerárquico es dendograma. Como este gráfico es de mucha ayuda para determinar el punto óptimo de corte lo he representado, agrupando en el número de clústers que parecía mejor según *k-means*, pero únicamente para ayudar al ojo, ya que lo represento, entre otras cosas, para dilucidar ese valor. Su aspecto es el siguiente:



El dendograma no deja claro cual es el mejor punto de corte. Si que parece que hay un par de conjuntos significativamente menores que el resto, pero entre 4 y 6 el punto óptimo de corte no queda claro. Si mostramos el gráfico que hicimos para *k-means* para este algoritmo veremos:



Donde es interesante notar que aunque la disposición de los conjuntos se conserva, el conjunto inferior es bastante menor que cuando usé *k-means* y no es hasta la agrupación en 6 conjuntos cuando vemos dividirse el conjunto de la esquina superior izquierda. Como el proceso que describimos para *k-means* en la esquina superior derecha se repite aquí, voy a seleccionar 5 como el número óptimo de conjuntos. Y como la agrupación jerárquica me parece más limpia es también la que voy a usar.

5.2 Predicción por grupos

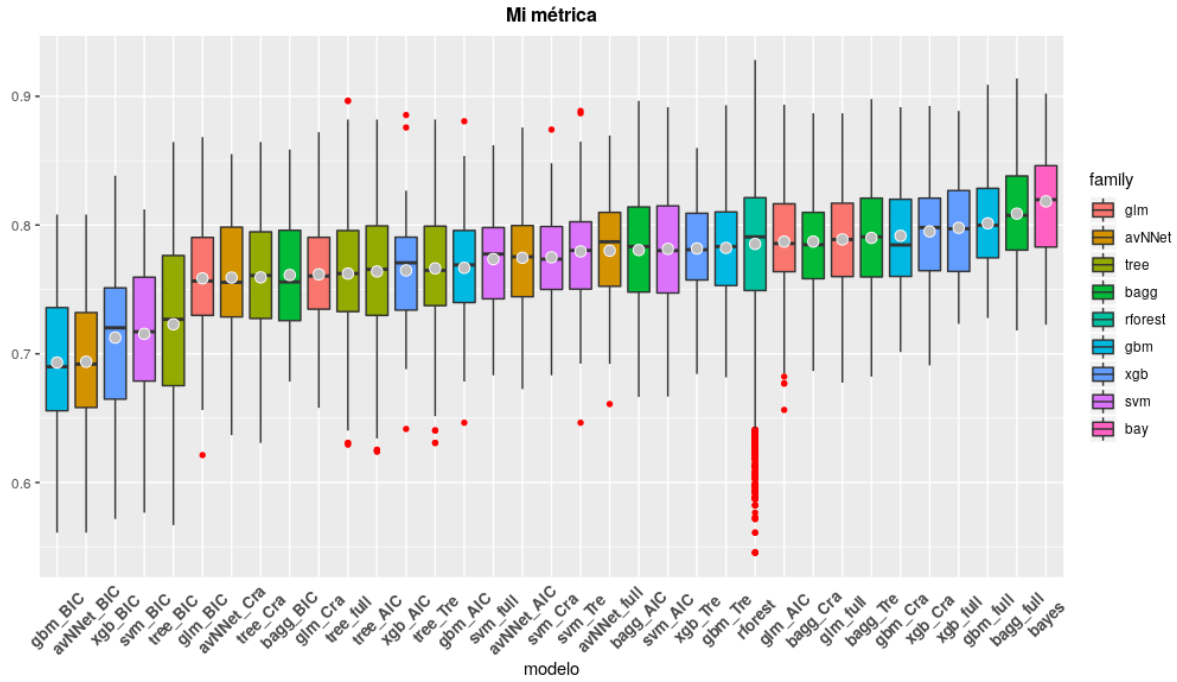
Finalmente lo que he hecho ha sido sobre mis datos agrupados seleccionar los 5 modelos que mejor precisión presentaban (en general no había cambios entre los primeros pero si entre los 2 o 3 últimos) y **usando las probabilidades** (antes usé las predicciones en binario) predecir por la media de lo que dijeron estos 5 modelos, y la media ponderada, esta vez ponderando de forma proporcional a la precisión del modelo sobre ese conjunto. Tenemos por lo tanto dos nuevas predicciones que subir, únicamente difieren en 2 filas, así que cruzo los dedos.

No hubo suerte, ambas presentan el mismo *score* **77'511%**. Por lo menos sé que de las dos observaciones que cambian de una a otra cada una acierta una.

5.3 *Naive* Bayes

Llegados a este punto he barajado multitud de opciones: repetir el mejor modelo pero para el *subset* de datos asociado a cada clúster, comparar las predicciones (ahora mismo tengo 38) y hacer trampas, etc. Pero la última opción que voy a barajar es la de coger el conjunto completo de datos, con las probabilidades de cada modelo y los conjuntos del *clustering* como factor, y realizar un último modelo.

El modelo que he realizado es un modelo *naive* bayesiano que vimos en un modulo reciente como usarlo, dió buenos resultados y está implementado en ***caret***. El proceso realizado es esencialmente el mismo solo que con un *set* de datos mayor y los resultados, para variar, no han mejorado. **78'947%** esa es mi puntuación final, pese a que el modelo si se colocaba a la cabeza en el gráfico.



El modelo que ha salido vencedor es la regresión logística, aunque no lo represente como tal el gráfico. Mientras que si no dispusiera de unos datos "futuros", cuyos valores desconocía, posiblemente estaría concluyendo el trabajo entusiasmado, ya que desde ***glm_AIC*** si observamos mejora en el gráfico, el hecho de haber elegido un concurso (de los sencillos) de Kaggle para el trabajo ha sido educador. No hay que obsesionarse con mejorar los parámetros de los algoritmos, pues, en general, el margen de mejora del *score* es pequeño. El análisis y la manipulación de los datos son lo que merecen mayor atención.

6 El código

Todo el código asociado a este trabajo se encuentra disponible en el repositorio de GitHub y se explica de la forma siguiente

6.1 Estructura de carpetas

```
root -> all the code files, the data, and main/this document
|___img -> all the images used in this document
|___mod -> all the models developed during the exercise
|___sub -> all the submissions made in Kaggle
|___tab -> all the tables used in this document
```

6.2 Relación ficheros de codigo

- **V01** → 2 Limpieza de datos, 3 Selección de variables y 4.1 Regresión Logística
- **V02** → 4.2 Perceptron Multicapa (MLP)
- **V03** → 4.3 Árboles de decision
- **V04** → 4.5 *Bagging* y *Random Forest*: Únicamente la parte de *bagging*
- **V05** → 4.6 *Bagging* y *Random Forest*: Únicamente la parte de *random forest*
- **V06** → 4.6 *Gradient Boosting*
- **V07** → 4.7 *XGBoost*
- **V08** → 4.8 Máquinas de vectores de soporte (SVM)
- **V10_predGenerator** → Tras detectar varios errores usé los modelos para regenerar los ficheros.
- **V11** → 5.1 *Clustering* de observaciones, 5.1.1 *K-Means* y 5.1.2 Jerárquico
- **V12** → 5.2.1 *Naive* Bayes